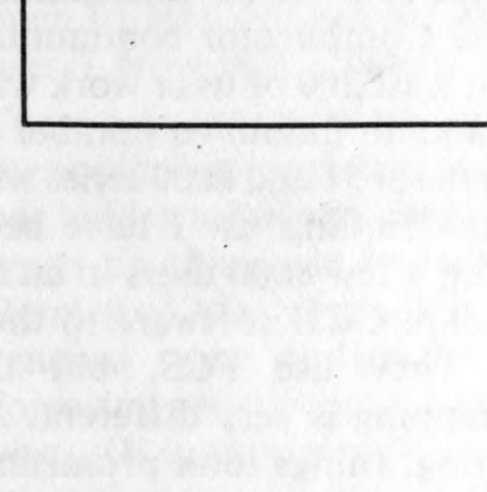
Whence did the wond'rous mystic art arise,
Of painting Speech, and speaking to the eyes?
That we by tracing magic lines are taught,
How to embody, and to colour thought?



COLORCUE VOLUME VI, NUMBER 3 MAY/JUNE 1984

Colorcue

VOLUME VI, NUMBER 3 MAY/JUNE 1984

CONTENTS

COVER: Text set by Carl Remley in pen and ink. Unknown source, quoted from CompUKolour.

EDITOR: JOSEPH NORRIS COMPUSERVE: 71106, 1302

COLORCUE is published bi-monthly. Subscription rates are US$18/year in the U.S., Canada, and Mexico (via First Class mail), and US$30 elsewhere (via Air Mail). All editorial and subscription correspondence should be addressed to COLORCUE, 19 West Second Street, Moorestown, NJ 08057, USA. (609-234-8117) Every article in COLORCUE is checked for accuracy to the best of our ability but is not guaranteed to be error free.

"Welcome to the Sourcebook...

You will find we couldn't deliver all we promised for this issue of Colorcue. In spite of written requests for information, mailed in January, many respondents sent materials at press time—too late. Others replied in such a vague way that nothing less than a divining rod could have decyphered the meaning. Still others used a handwriting (with faint pencil) that could only defeat any mortal interpretation. The fault is not to be entirely externalized, however. We simply bit off more than we (and our bank account) could chew. Not to worry. The promise will be fulfilled in time. Meanwhile, I hope this issue is as interesting to you as it has been to me.

Welcome to those of you who have rejoined us, and to those who are new to the CCII by virtue of purchasing used computers. You are discovering that we have a lot to offer (at a reasonable price) in the Compucolor community. There are at least 200 of us at work with the CCII, and an unknown number of others with the 3651 and 8000 series who are just now finding us. I have been working with a few 8000 users in an attempt to adapt CCII software to their machines. They use FCS, but the memory mapping is very different. As of this writing, things look promising. This might mean a somewhat revitalized software market for Com-tronics, Bill Greene and anyone else still willing to try.

The highlight of my Compucolor experiences since the last issue was a visit to Rochester and the user group there. Imagine, if you can, eleven or so owners, all gathered together in the same place. They've been doing it steadily for years, and I imagine they have an inadequate appreciation for the comradeship and mutual support CHIP gives them. The topic for the evening was FORTH, one in a series of talks by

.....ah...that is....Part 1.''

Jim Minor on the FORTH language. I delighted in putting a ''face'' on Gene Bailey, Doug Van Putte, Rick Taubold, Joseph Charles, Jim Minor, David Suits, and all the other names I've known only in print. Dr. and Mrs. Suits were my gracious hosts for a night. David and his wife have started a new magazine for the NEC APC computer, called NEXUS. It has a familiar format, and the contents remind one of early CCII days—starting all over again. It is an appropriate computer for David—spectacular graphics, rather complicated operating system, and a challenge for him for a few years, I imagine. My thanks to the CHIP group for letting me sit in, and I hope to see you all again soon. Wouldn't it be wonderful if CHIP could arrange a grand reunion for all of us? How about an International CCII Conference for a weekend in Rochester, open to all Compucolor and Intecolor users? Now that's entertainment!

There are some new authors coming on the scene. I guess rightly that our group has hidden talents. Witness the beautiful cover on this issue by subscriber Carl Remley of Mississippi. His subscription renewal came written in calligraphic form and I was prompt to enlist his services. (It is appropriate that we express more than our computer interest in these pages.) The text is reprinted from the first issue of COMPUKOLOR, of the user group in the United Kingdom. (I do not have the source of this quotation. Does anyone know it?

So, on to the first part of the SOURCEBOOK. Rest assured that all submitted materials will be published, and all promises fulfilled. We look forward to your new materials as we continue our sixth year of publication. Thanks to the many of you who have sent notes of appreciation, and... keep the cards and letters coming.

John.

A Pascal for the CCII

Part II.

Doug Van Putte
18 Cross Bow Drive
Rochester, NY 14624

In Part I, a subset of Pascal, called Tiny-Pascal, was introduced through the fig-Forth language. Since Tiny-Pascal uses the facilities of Forth, such as editing, compiling and disk handling, Part I told how to obtain and implement both Forth and Tiny-Pascal on the CCII. In addition, Part I presented the Forth editor commands required to enter, edit, store, compile, and run Tiny-Pascal programs.

We will proceed by introducing some programming techniques that relate to the structure of a Pascal program. Pascal is designed as a ''block-structured'' language. The blocks are the basic building units of the program, so it is important to have a good conception of them. Think of blocks as logical subdivisions of program functions. Writing a program to bake a cake? Then design blocks to handle each part of the job: read the recipe, measure the ingredients, mix the ingredients, and perform the baking. The effectiveness of this block structure might not seem apparent in a short program, but in very complex programs it provides a clear advantage over Basic command structures.

A block structure enables the distribution of a complex program task into simple, smaller program tasks, which may be tested and debugged in themselves as they are created. Then, as the blocks are connected together, debugging will be limited to problems in the interaction among the blocks, greatly reducing debugging time. This procedure may be followed in Basic as well, if you exercise self-discipline, but the structure of Pascal syntax dictates these good programming practices naturally. A properly-written Pascal program is more readily accessible by other readers, and years from now it will reveal itself with the same clarity it had at its conception. A typical block-structured outline of a program is shown in Listing 1. The middle column contains the actual program code. The first and third columns are comments.

As the listing begins, we see a program name (PROGRAM DUMPCAKE). We see a declaration of a constant OVENTEMP, and a declaration of the variables CUP which is to be an integer number, and TEASPOON which is also to be an integer. In Pascal, constants and variables must be declared (identified) before they can be referenced by the program. This treatment is distinctly different from many versions of Basic, where one can assign a non-dimensioned numeric or string variable anywhere in the body of a program without regard to its type (real or integer).

Next we see a procedure, PROC RECIPE (procedure ''recipe''), and a function, FUNC PROPORTION (function ''proportion'') specified. RECIPE might tell how to mix the ingredients; PROPORTION might describe how to proportion the mixture for various sizes of cake. These are ''subroutines'' which can be ''called'' by the main program. The declaration of variables and the specifying of procedures and functions are placed at the head of the program so they may be ''seen'' by the program before it begins execution. (This is much like reading data statements in Basic, before using the data.) You will notice that both the procedure and function portions may be ''sub-programs'', of several lines, each marked with a ''BEGIN'' and ''END'' to define the scope of each.

Now we approach the main body of the program which schematically represents several groups of coded instructions. Some elements of the group are ''nested'' (like FOR....NEXT

statements in Basic) and others are sequential. We use indentation in the listing to help the eye separate the groupings according to the sequence in which they perform.

Notice the punctuation after the END statements. Those END statements that are within the body of the program are followed by a semi-colon to indicate that the program continues. The final END statement is followed by a period (.) to indicate the conclusion of the program. Of course a program may have a wide variety of these BEGIN...END sections in all combinations.

The listing also contains statements set within the reverse backslash marks, which are comment sections (like REM statements in Basic.) Pascal differs from Basic in that comments may be placed both before and after a code statement. The compiler will find the code statement and extract it.

Let's see how to specifically design a Tiny-Pascal program through the FORTH language. We will do this by writing a program to compute the area in a 2 by 4 rectangle (Listing 2). The spaces and punctuation are required, as shown, both for FORTH and Pascal syntax. Since we must first proceed from FORTH, we will follow a FORTH convention by placing the name of our program in parentheses on the first line. Parentheses are "delimiters" (comment markers) in FORTH. This will be followed by the FORTH words "PASCAL", which invokes the Tiny-Pascal interpreter, and "DECIMAL", which tells FORTH that our numerical data will be entered in decimal format. (We could have said "HEX" instead, if we meant to use hexadecimal numbers.) This is all we need to do to satisfy FORTH's requirements.

Now we write line 2 for Pascal, the Pascal word PROGRAM followed by our program name RECTANGLE AREA. Observe the terminating semi colon. Next we can see that two con stants are declared (LENGTH and WIDTH) and values assigned to them. In choosing the names of these constants, and the names of all the elements of our program, such as the program name, variables, procedures and functions, Pascal allows us to use both letters and digits to a length of 31 characters. This permits the names to be precisely descriptive.

A variable is declared next (AREA) and it is declared as an integer value. Notice the colon following the declaration of a variable. Now the "main" program begins and we equate AREA to the product of LENGTH and WIDTH. NEWLINE advances control to the beginning of a new line, and WRITE, with its printed prompt, prints the product on the CRT. A BEGIN statement and END statement bracket this main program block.

While this program could be duplicated with a single statement in Basic, it serves as an example of Pascal.

LISTING 1. Schematic outline of a Pascal Program.

```
\PASCAL PROGRAM\      PROGRAM DUMPCAKE;                \MAIN BLOCK HEADING\

\DECLARATIONS\        CONST OVENTEMP = 350;            \BEGIN DECLARATION BLOCK\

                      VAR CUP: INTEGER;

                          TEASPOON: INTEGER;           \END DECLARATION BLOCK\

  \PROCEDURES\        PROC RECIPE;                     \BEGIN PROCEDURE BLOCK\

                        BEGIN
                          ----------
                          ----------
                          ----------
                          ----------
                        END;                           \END PROCEDURE BLOCK\

  \FUNCTIONS\         FUNC PROPORTION;                 \BEGIN FUNCTION BLOCK\

                        BEGIN
                          ----------
                          ----------
                          ----------
                        END;                           \END FUNCTION BLOCK\

\MAIN PROGRAM\        BEGIN                            \BEGIN MAIN BLOCK 1\
                        ----------
                        ----------
                        BEGIN                          \START BLK 2\
                          ----------
                        END;                           \END BLK 2\

                        BEGIN                          \START BLK 3\
                          ----------
                          BEGIN                        \START BLK 4\
                            ----------
                            ----------
                          END;                         \END BLK 4\
                          ----------
                        END;                           \END BLK 3\
                        ----------
\END OF PROGRAM\      END.                             \END MAIN BLOCK 1\
```

As the program becomes more complicated, Basic soon loses it's advantage of brevity, and the clarity of Pascal emerges. Among the peculiarities of punctuation, notice the semicolon that terminates each statement line, the ":=" used to establish an equate, the use of '.....' to delimit text in the WRITE statement, and "-" to designate printing the value of AREA. NEWLINE is the only "cursor positioning" statement in Tiny-Pascal, and gives direction to commence subsequent printing on the next line.

You may enter the program in Listing 2 on a blank FORTH screen using the editor. Check it carefully for spaces and punctuation. FLUSH the screen to disk. To compile the program, type [n] LOAD, where [n] is the number of the screen containing this program. From this point, you may "run" the program by typing RECTANGLEAREA.

Our program would be more functional and more interesting if we could enter new parameters for LENGTH and WIDTH from the keyboard. The READ statement in Pascal makes this possible, and operates like the "INPUT" statement in Basic. To prompt an input, the WRITE statement is used, followed by READ. We will have to change LENGTH and WIDTH from constants to integer variables, and insert a few statement lines just after the BEGIN in Listing 2. The changes are shown in Listing 3. Add them to the FORTH screen with the editor, and proceed as before to compile and run the amended program. Press RETURN after entering "length", and again after entering "width."

If you want to try some more adventurous programing, here are some other arithmetic operations available in Tiny-Pascal. "+", "-", and "*" are used as in Basic. "DIV" is used in Tiny-Pascal for integer division. The "/" symbol is reserved for real number division, and is not supported by Tiny-Pascal. We will be looking at other Pascal constructs, such as IF...THEN, WHILE...DO, FOR...DOWNTO, CASE...OF, and REPEAT...UNTIL. These operators, along with some exploration into PROC and FUNC organization will provide many possibilities for practical, interesting programs. Good luck, and good learning!□

Listing 2. A working program to calculate and print area.

```
( RECTANGLEAREA ) PASCAL DECIMAL    \FORTH commands\

PROGRAM RECTANGLEAREA;              \computes and prints\
                                    \area=l*w\

CONST                               \Declare constants\
  LENGTH = 4;
  WIDTH = 2;
VAR                                 \Declare variable\
  AREA: INTEGER;

BEGIN                               \Main program\

  AREA := LENGTH * WIDTH;
  NEWLINE;
  WRITE ( 'The area is ', #AREA);

END.                                \End of program\
```

Listing 3. Changes to RECTANGLEAREA to permit keyboard entry.

```
\Change declaration of these labels to variables\

   LENGTH, WIDTH: INTEGER;

\Add these lines after BEGIN\

   NEWLINE;                  \Print crlf\

   WRITE ( 'Enter the rectangle length & width ' );

   NEWLINE;

   READ ( #LENGTH,#WIDTH );  \Read two variables\
                             \from keyboard    \
```

Errata: In Part I, the screen editor command to copy one screen to another was incorrect. "COPY [num1] [num2]" should be corrected to read "[num1] [num2] COPY"

ROM TABLES FROM THE SYSTEM LISTINGS v6.78, v8.79, v9.80-3

This table is compiled from the system listing of three software versions of FCS. (- - -) = NA (not applicable); (• • •) indicates that the address is the same as in the preceeding column. Frequently used labels appear in boldface.

LABEL	v6.78	v8.79	v9.80
A7ON	38E8	0331	0300
ACRTSP	0036	• • •	• • •
ADDU	2144	1AE9	1B76
ADHLA	**3518**	**194E**	• • •
ADJTKS	- - -	- - -	1C43
AESCTB	000B	• • •	• • •
ANHD	351D	1953	• • •
ASCPL	3DFB	0859	07FB
AUCNT	81B3	• • •	- - -
AUTOX	0058	1F3E	• • •
B2HEX	**33AA**	**17E0**	• • •
B7ON	3A19	0462	0431
BA7OF	3946	038F	035E
BARTX	3D5F	07BD	075F
BARTY	3D57	07B5	0757
BARTZ	3D51	07AF	0751
BARXM	3C13	066A	060C
BARYM	3C42	0699	063B
BARYM1	- - -	- - -	063E
BARYM2	- - -	- - -	064A
BASEX	**0055**	**1F3B**	• • •
BASFL	**81F1**	• • •	• • •
BASICE	**0046**	**1F2C**	• • •
BASICI	0052	1F38	• • •
BASICW	0040	1F26	• • •
BASORG	0040	1F26	• • •
BASOUT	**0033**	• • •	• • •
BAUD	**0005**	• • •	• • •
BC01	35B2	19EB	• • •
BC2BK	35A8	19DE	• • •
BCCIX	3A05	044E	041D
BCHK	3292	16C8	• • •
BCHK1	32BB	16F1	• • •
BCRSX	3A2A	0473	0442
BCRSY	3A37	0480	044F
BEGEX	**0038**	• • •	• • •
BEGIN	**3768**	**01B6**	• • •
BEGOT	3A59	04B0	047F
BEL	**3AC3**	**051A**	**04E9**
BEL1	- - -	- - -	04EE
BFILL	81D0	• • •	• • •
BFSX	- - -	- - -	001F
BHLAD	**81D4**	• • •	• • •
BK2BC	35BA	19F0	• • •
BKCOL	3928	0371	0340
BLIND	**3A09**	**0452**	**0421**
BLINK	393F	0388	0357
BRAKE	3AB6	050D	04DC
BRATE	3A09	0452	0421
BRATX	3A6F	04C6	0495
BREAK	**003B**	• • •	• • •
BRTR	- - -	- - -	0010
BRTRY	80E0	• • •	- - -
BRTX1	- - -	- - -	049D
BS01	35ED	1A23	1E4E
BS02	35F1	1A27	1E52
BS03	- - -	- - -	1EB2
BS04	365F	1A95	1EFA
BS10	237E	1D01	- - -
BS11	2389	1D0C	- - -
BS12	239D	1D20	- - -
BS13	236B	1CEE	- - -
BSB01	35C7	19FD	1E26
BSB02	- - -	- - -	1E92
BSB03	- - -	- - -	1E98
BSB04	- - -	- - -	1E84
BSB05	- - -	- - -	1EBD
BSB06	- - -	- - -	1EC9
BSB07	- - -	- - -	1ED1
BSB08	3652	1A88	1EED
BSDHU	- - -	- - -	19DE
BSTR	33E9	181F	• • •
BUCNT	81B4	• • •	- - -
BUFP	**8047**	• • •	• • •
BXLOP	3C30	0687	0629
BYLOP	3C67	06BE	0660
BYLOP1	- - -	- - -	066F
C3C9	- - -	- - -	011F
C3C95	- - -	- - -	012A
CARET	000D	• • •	• • •
CARR	3B4E	05A5	056C
CBC	30F5	152B	• • •
CBC1	3108	153E	• • •
CBC2	3127	155D	• • •
CCI	3A09	0452	0421
CCIX	3A01	044A	0419
CD03	2215	1BC1	- - -
CD04	2219	1BC5	- - -
CDHD	211C	1AC1	- - -
CDK	3695	004A	- - -
CDMK	002E	• • •	• • •
CDNM	3691	0046	- - -
CDNU	3693	0048	- - -
CDMK	- - -	- - -	002E
CDRSET	218B	1B30	- - -
CDSEC	3694	0049	- - -
CEN01	2AA5	0F48	0F45
CENTR	2A9E	0F41	0F3E
CHAIN	3A09	0452	0421
CHDEL	000E	• • •	• • •
CHDLR	2EFC	1332	• • •
CHPLO	39FD	0446	0415
CHTIM	001C	• • •	• • •
CKEND	26E7	0B8A	• • •
CLOSE	**2F26**	**135C**	• • •
CLSEQO	**3136**	**156C**	• • •
CLX	2859	0CFC	• • •
CMASK	81E0	• • •	• • •
CMDTMP	- - -	- - -	81B1
CMPDH	**3453**	**1889**	• • •
CMPHD	**344D**	**1883**	• • •
CMT1	2AB4	0F57	0F54
CMT2	2ABF	0F62	0F5F
CMTAB	257A	0A58	08D4
CODE	3996	03DF	03AE
CODE2	39A2	03EB	03BA
COLFL	**81E6**	• • •	• • •
COLOR	3907	0350	031F
COLW	392C	0375	0344
COMND	0004	• • •	• • •
COMOF	393B	0384	0353
COMON	393A	0383	0352
COP00	2B20	0FC3	• • •
COP01	2B38	0FDB	• • •
CPLOX	3E03	0861	0803
CPYDV	2C77	10AD	• • •
CR	3872	02BB	028C
CR1	24B3	1E36	- - -
CR2	24B8	133B	- - -
CR3	24BF	1E42	- - -
CR4	24C3	1E46	- - -
CRATE	**81E2**	• • •	• • •
CRC	247D	1E00	- - -
CRC1	8043	• • •	- - -
CRC2	8044	• • •	- - -
CRCX	249F	1E22	- - -
CRET	3B56	05AD	0574
CRLF	338B	17C1	• • •
CRSLT	38F0	0339	0308
CRSRT	38B5	02FE	02CD
CRSUP	38F6	033F	030E
CRSUP1	- - -	- - -	0316
CRSXY	3809	0452	0421
CRSY	388D	02D6	02A7
CRT0	0060	• • •	• • •
CRT1	0061	• • •	• • •
CRT2	0062	• • •	• • •
CRT3	0063	• • •	• • •
CRT4	0064	• • •	• • •
CRT5	0065	• • •	• • •

LABEL	v6.78	v8.79	v9.80
CRT6	0066	• • •	• • •
CRTCHIP	0060	• • •	• • •
CRTMO	25B7	01A7	• • •
CRTMSG	25C6	0123	0040
CRTR	0003	• • •	• • •
CRTRAM	81AF	• • •	• • •
CRTRY	80E2	• • •	81AE
CRTSET	37C0	020F	01EB
CRTST1	- - -	- - -	01F0
CRTUBE	396B	03B4	0383
CRXDA	006C	007C	• • •
CRYDA	006D	007D	• • •
CTRK0	81B1	• • •	- - -
CTRK1	81B2	• • •	- - -
CTWO	2C69	109F	• • •
CTYP	- - -	- - -	109A
CUCNT0	81B5	• • •	- - -
CUCNT1	81B6	• • •	- - -
CURSO	3A0B	0454	0423
D1	358A	19C0	• • •
D2	359F	19D5	• • •
D5CNT	- - -	- - -	6F84
D8CNT	- - -	- - -	6F83
DATAM	005A	• • •	- - -
DBF	811D	• • •	• • •
DBFE	819D	• • •	• • •
DBLK	811D	• • •	• • •
DDFHD	- - -	- - -	1BAB
DDFHV	- - -	- - -	1B05
DDFNM	- - -	- - -	1B08
DDFNU	- - -	- - -	1B0A
DDFFC	- - -	- - -	1B0B
DDFTK	- - -	- - -	1B0C
DDMHD	- - -	- - -	1B23
DDMHV	- - -	- - -	1AED
DDMNM	- - -	- - -	1AF0
DDMNU	- - -	- - -	1AF2
DDMRT	- - -	- - -	1AF5
DDMSC	- - -	- - -	1AF3
DDMSF	- - -	- - -	1AF6
DDMTK	- - -	- - -	1AF4
DEL00	29B5	0E58	• • •
DEL01	29D6	0E79	• • •
DEL06	2A2B	0ED0	• • •
DEL07	2A3C	0EDF	• • •
DEL08	2A44	0EE7	• • •
DEL09	2A6F	0F12	• • •
DEL10	2A8A	0F2D	• • •
DELER	2A9B	0F3E	0F3B
DELTA	000F	0014	- - -
DEV00	2996	0E39	• • •
DFDHD	- - -	- - -	1BDD
DFDHV	- - -	- - -	1B11
DFDNM	- - -	- - -	1B14
DFDNU	- - -	- - -	1B16
DFDSC	- - -	- - -	1B17
DFDSF	- - -	- - -	1B1A
DFDTK	- - -	- - -	1B18

LABEL	v6.78	v8.79	v9.80
DFDV	80F0	• • •	• • •
DFH	- - -	- - -	1B03
DFUN	80F2	• • •	• • •
DIG	3476	18AC	• • •
DIP	2D80	11B6	• • •
DIPS	0006	• • •	• • •
DIR00	2791	0C34	• • •
DIR01	2799	0C3C	• • •
DIR02	27C5	0C68	• • •
DIR03	2816	0CB9	• • •
DIR04	282A	0CCD	• • •
DIR05	27D9	0C7C	• • •
DISPCK	81BC	• • •	• • •
DIVHD	3581	19B7	• • •
DMDHD	- - -	- - -	1B52
DMDHV	- - -	- - -	1AF9
DMDNM	- - -	- - -	1AFC
DMDNU	- - -	- - -	1AFE
DMDRT	- - -	- - -	1B01
DMDSC	- - -	- - -	1AFF
DMDSF	- - -	- - -	1B02
DMDTK	- - -	- - -	1B00
DOWN	3A90	04E7	04B6
DSBUF	6000	• • •	• • •
DSBUFS	1000	• • •	• • •
DSEC	226A	1C10	1CE7
DUP00	2B93	0120	- - -
DUP02	2BD9	- - -	- - -
DUPLX	81DD	• • •	• • •
DX01	- - -	- - -	1C74
DX02	- - -	- - -	1C8F
DX03	- - -	- - -	1CC6
DX04	- - -	- - -	1CE0
DX05	- - -	- - -	1CE3
DX10	2132	1AD7	- - -
DX11	2149	1AEE	- - -
DX12	2159	1AFE	- - -
DX13	2169	1B0E	- - -
DX13A	216E	1B13	- - -
DX14	21AB	1B50	- - -
DX14A	21C3	1B68	- - -
DX15	21D1	1B76	- - -
DX16	21E6	1B8B	- - -
DX17	21E6	1B8B	- - -
DX18	21F0	1B95	- - -
DXCM1	- - -	- - -	1C18
DXCM2	- - -	- - -	1C26
DXX	2201	1BA6	1C67
DZ01	- - -	- - -	1BC7
DZ02	- - -	- - -	1BD3
DZ03	- - -	- - -	1BD8
DZ04	- - -	- - -	1C0B
DZ05	- - -	- - -	1C13
DZ10	- - -	- - -	1C1F
DZ20	- - -	- - -	1C38
EARLN	3AF8	054F	051E
EARLN1	- - -	- - -	0527
EARS	3839	0282	0253

LABEL	v6.78	v8.79	v9.80
EARS0	- - -	- - -	0255
EARS1	- - -	- - -	025A
EBLF	004B	0045	• • •
EBLK	0009	• • •	0003
ECFB	000F	• • •	0006
ECOP	0045	003F	• • •
EDCS	0015	• • •	0009
EDEL	003C	• • •	003C
EDEN	0024	• • •	- - -
EDFN	- - -	- - -	0024
EDIR	000C	• • •	000C
EDRF	002D	• • •	• • •
EDFY	0012	• • •	- - -
EDUP	003F	- - -	- - -
EFCS	3372	17A8	17A6
EFNF	002A	• • •	• • •
EFRD	0030	• • •	• • •
EFWR	0033	• • •	• • •
EHCS	- - -	- - -	000C
EIVC	0003	• • •	• • •
EIVD	001E	• • •	• • •
EIVF	0003	• • •	000F
EIVP	000F	• • •	• • •
EIVT	0048	0042	• • •
EIVU	0006	• • •	0012
ELIN	334E	1784	1782
ELINE	3AEC	0543	0512
EMDV	0018	• • •	• • •
EMEM	0018	• • •	0015
EMESS	**262D**	**0AD6**	• • •
EMFN	0015	0015	0015
EMVN	0006	• • •	• • •
EMVR	001B	• • •	• • •
ENSA	0021	• • •	• • •
ENVE	0012	• • •	• • •
EOFOK	2947	0DEA	• • •
EPRM	- - -	- - -	4000
ERALP	3851	029A	026B
ERAS	**2B8D**	**1030**	**- - -**
ERASE	3836	027F	0250
ERS1	3631	1A67	0A3D
ERS2	3637	1A6D	0AF3
ERSYN	26EB	0B8E	0B8E
ERSZ	0039	• • •	0039
ES01	- - -	- - -	1D2F
ES02	- - -	- - -	1D53
ES03	- - -	- - -	1D5A
ES04	- - -	- - -	1D6D
ES05	- - -	- - -	1D76
ESCAP	3A09	0452	0421
ESCAX	3AAA	0501	04D0
ESCCRT	81BF	• • •	• • •
ESCD	**32C9**	**16FF**	• • •
ESCDG	32D1	1707	• • •
ESCG	32BE	16F4	• • •
ESCTB	36D8	008D	• • •
ESEC	2354	1CD7	1D22
ESIZ	0042	- - -	- - -
ESKF	000C	• • •	0018

LABEL	v6.78	v8.79	v9.80
ESYN	0009	• • •	• • •
ETYP	- - -	- - -	005A
EVEN	- - -	- - -	1DE6
EVFY	001B	• • •	• • •
EVOV	0027	• • •	• • •
EWRF	- - -	- - -	001E
EWSZ	0036	• • •	• • •
EXDHV	- - -	- - -	1B1D
EXE00	- - -	- - -	1030
EXE05	- - -	- - -	1A30
EXE10	- - -	- - -	1A61
EXE100	- - -	- - -	1ABA
EXE20	- - -	- - -	1A69
EXE30	- - -	- - -	1A6C
EXE40	- - -	- - -	1A76
EXE50	- - -	- - -	1A8A
EXE60	- - -	- - -	1A97
EXE70	- - -	- - -	1AA3
EXE80	- - -	- - -	1AA8
EXE90	- - -	- - -	1AB8
EXH	- - -	- - -	1B1B
EXTBF	81D6	• • •	• • •
EXTIN	0001	• • •	• • •
EXTOT	0007	• • •	• • •
F1	25EF	0A98	• • •
F2	25F2	0A9B	• • •
F3	25F9	0AA2	• • •
F4	2607	0AB0	• • •
F5	- - -	- - -	0AB4
FATR	**80F8**	• • •	• • •
FAUX	**810F**	• • •	• • •
FBLK	**8115**	• • •	• • •
FBUF	**8117**	• • •	• • •
FCS	**25EC**	**0A95**	• • •
FCSEM	**262A**	**0AD3**	• • •
FCSEX	**2622**	**0ACB**	• • •
FCSFL	**81E1**	• • •	• • •
FCSORG	25B7	- - -	- - -
FCSOUT	**3379**	**17AF**	• • •
FCSX	3301	1737	• • •
FDBK	**810D**	• • •	• • •
FDEN	**810E**	• • •	• • •
FDH	- - -	- - -	1B0F
FDRV	**8114**	• • •	• • •
FEED	3B3D	0594	055B
FERS1	2648	0AF1	• • •
FERS2	265A	0B03	• • •
FFCN	8113	• • •	• • •
FG0	226C	1C12	- - -
FG1	226E	1C14	- - -
FG2	2272	1C18	- - -
FG3	227F	1C25	- - -
FG4	2286	1C2C	- - -
FHAN	**8111**	• • •	• • •
FILL	00FF	• • •	- - -
FLAD	**8108**	• • •	• • •
FLBC	**8107**	• • •	• • •
FNAM	**80F9**	• • •	• • •
FNEW	0001	• • •	• • •
FPB	**807F**	• • •	• • •
FPBE	811D	• • •	• • •
FPBP	80F3	• • •	• • •
FPROM	0080	- - -	- - -
FPTR	**811B**	• • •	• • •
FREE	3A0A	0453	0422
FREEX	3A0A	0453	0422
FSAD	**810A**	• • •	• • •
FSBK	**8103**	• • •	• • •
FSIZ	**8105**	• • •	• • •
FTYP	**80FF**	• • •	• • •
FULL	3A4E	04A5	0474
FVER	**8102**	• • •	• • •
FXBC	**8119**	• • •	• • •
G01	3504	193A	• • •
GAR1	3258	168E	• • •
GAR2	3272	16A8	• • •
GAREC	**3257**	**168D**	• • •
GB1	3224	165A	• • •
GCMA	3488	18BE	• • •
GCTRK	256A	1EEB	- - -
GCUCNT	2570	1EF1	- - -
GDATAM	24C8	1E4B	- - -
GDRET	271E	0BC1	• • •
GETBC	37FC	024A	0226
GETBC0	- - -	- - -	0222
GETTO	2C0C	1042	• • •
GH1	22DB	1C5E	- - -
GH2	22E2	1C65	- - -
GH3	22E9	1C6C	- - -
GH4	22F3	1C76	- - -
GM01	2CCD	1103	• • •
GM02	2CD0	1106	• • •
GM03	2CBD	10F3	• • •
GMPRM	2CA4	10DA	• • •
GN1D	34E7	191D	• • •
GN1Z	34E4	191A	• • •
GN2D	34F9	192F	• • •
GN2Z	34F6	192C	• • •
GNDE	2D86	11BC	• • •
GNDE0	- - -	- - -	013E
GODBk	2FB5	13EB	• • •
GTBYT	**322C**	**1662**	• • •
GXOUT	2564	1EE5	- - -
HALF	3A4C	04A3	0472
HANADD	- - -	- - -	80E3
HANER	35FC	1A32	1E5D
HDCNT	- - -	- - -	6F85
HDVCT	368E	0043	1AEB
HER1	22A5	1C4B	- - -
HERR	- - -	- - -	1DDE
HEX	81B8	• • •	• • •
HOLD	- - -	- - -	002A
HOME	386E	02B7	0288
HOMEX	- - -	- - -	1CCB
HRTR	001E	• • •	- - -
IDEV	368B	0040	0A88
IDM	0055	• • •	- - -
INCXY	3D2C	078A	072C
INCXY0	- - -	- - -	072A
INI00	2721	0BC4	• • •
INI01	2761	0C04	• • •
INI02	2766	0C09	• • •
INI03	2768	0C0B	• • •
INIBAS	37BD	01EB	01E5
INITAD	0003	• • •	• • •
INPCRT	**81C5**	• • •	• • •
INPFL	81E3	• • •	• • •
INPTB	3728	00DD	• • •
INSEQO	**30E7**	**151D**	• • •
INVEC	3BC4	061B	05BA
INVY	3BE5	061B	05DE
INVY0	- - -	- - -	05DC
INVY1	- - -	- - -	05E2
IRBK	318E	15C4	• • •
IRBKI	3205	163B	• • •
IROLL	38C1	030A	02D9
IS1	2240	1BE6	- - -
IS2	224D	1BF3	- - -
IS3	2265	1C0B	- - -
IS4	2267	1C0D	- - -
ISEC	2235	1BE5	1CE5
ISERL	3974	03BD	038C
ISERX	3991	03DA	03A9
IUNT	368D	0042	0A8A
IVC	2604	0AAD	- - -
IWBK	318B	15C1	• • •
IWBKI	3202	1638	• • •
JMPD	2F01	1337	• • •
JMPHL	3FDD	0A3B	0F63
JUMP	**81E7**	• • •	• • •
KAP0	- - -	005C	• • •
KAP1	- - -	0079	• • •
KAP2	- - -	003B	• • •
KAP3	- - -	009F	• • •
KAP4	- - -	0021	• • •
KAP5	- - -	000C	• • •
KBCHA	**81FE**	• • •	• • •
KBDFL	**81DF**	• • •	• • •
KBR1	- - -	- - -	0378
KBRDY	**- - -**	**- - -**	**81FF**
KBREP	394F	0398	0367
KCHAR	**003E**	• • •	• • •
KEDEL	001D	• • •	• • •
KERDY	001E	• • •	• • •
KEYBD	3EB3	0911	• • •
KEYCO	002B	• • •	• • •
KEYOT	3A53	04AA	0479
KEYOT1	- - -	- - -	047D
KEYTST	- - -	- - -	0024
KEYTEST	**0024**	• • •	• • •
KTAB	3FDE	0A3C	08B8
L001	2D94	11CA	• • •
L002	2D82	11D8	• • •
L005	2DC9	11FF	• • •

LABEL	v6.78	v8.79	v9.80
L006	2E16	124C	• • •
L007	2E30	1266	• • •
L008	2E3D	1273	• • •
L009	2E40	1276	• • •
L010	2E4A	1280	• • •
L011	2E5C	1292	• • •
L012	2E75	12AB	• • •
L013	2E82	12B8	• • •
L014	2F14	134A	• • •
L017	2F60	1396	• • •
L023	2E7E	12B4	• • •
L025	2FAA	13E0	• • •
L1	32E9	171F	• • •
L106	2E15	124B	• • •
L2	32F7	172D	• • •
L3	332A	1760	• • •
L4	3334	176A	• • •
L5	3349	177F	• • •
LADSB	2F03	1339	• • •
LBYT	339B	17D1	• • •
LER1	294E	0DF1	• • •
LER2	2952	0DF5	• • •
LET	346A	18A0	• • •
LF	38BD	0306	02D5
LHXD	33A4	17DA	• • •
LINBF	**8046**	• • •	• • •
LINE	3B45	059C	0563
LINE1	- - -	- - -	0565
LKC	81E4	• • •	• • •
LL1	3E41	089F	0841
LL10	3EA4	0902	08A4
LL2	3E4D	08AB	084D
LL3	3E59	08B7	0859
LL4	3E65	08C3	0865
LL5	3E66	08C4	0866
LL6	3E7D	08DB	087D
LL7	3E80	08DE	0880
LL8	3E8C	08EA	088C
LL9	3E8D	08EB	088D
LLDA	28CD	0D70	• • •
LN5X	3897	02E0	02B1
LN5Y	389E	02E7	02B8
LO	**3392**	**17C8**	• • •
LOA00	2869	0D0C	• • •
LOCAL	3A4F	04A6	0475
LODG	3472	18A8	18A8
LOFL	**81F9**	• • •	• • •
LOL1	28C9	0D6C	• • •
LOL3	28E6	0D89	• • •
LOL4	28EA	0D8D	• • •
LOL5	291F	0DC2	• • •
LOL6	2936	0DD9	• • •
LOL7	293D	0DE0	• • •
LOLDA	288F	0D32	• • •
LOTO10	- - -	- - -	0138
LOTDUP	- - -	- - -	0130
LPJMP0	- - -	- - -	4803
LPJMP1	- - -	- - -	4806
LPJMP2	- - -	- - -	4809
LPJMP3	- - -	- - -	480C
LPJMP4	- - -	- - -	480F
LPJMP5	- - -	- - -	4812
LPJMP7	- - -	- - -	4818
LS1	3B93	05EA	0589
LS2	3B94	05EB	058A
LS3	3B9F	05F6	0595
LS4	3BA0	05F7	0596
LS5	3BAB	0602	05A1
LS6	3BAC	0603	05A2
LS7	3BBA	0611	05B0
LS8	3BBB	0612	05B1
LTIM	0B5A	• • •	• • •
LTNOR	347E	18B4	• • •
LTPEN	3B0A	0561	0530
LTYP	2D5A	1190	• • •
M1	356A	19A0	• • •
M2	357A	19B0	• • •
M8002	- - -	014A	• • •
MASK	0008	• • •	• • •
MCH01	2C2A	1060	• • •
MCH02	2C41	1077	1074
MCHNK	2C1A	1050	• • •
MDBLK	811E	• • •	• • •
MDDX10	- - -	- - -	1B3F
MDDX11	- - -	- - -	1B93
MDDX12	- - -	- - -	1BA3
MDH	- - -	- - -	1AF7
MFIOA	0000	• • •	• • •
MMEMO	- - -	018E	0175
MODE	0006	• • •	• • •
MOVDH	**343B**	**1871**	• • •
MOVHD	**3444**	**187A**	• • •
MOVXY	3D48	07A6	0748
MS1	2CF7	112D	• • •
MS150	81FD	• • •	• • •
MS2	2D04	113A	• • •
MS3	2D33	1169	• • •
MS4	2D42	1178	• • •
MST01	3496	18CC	• • •
MST02	34A3	18D9	• • •
MST03	34AB	18E1	• • •
MST04	34AD	18E3	• • •
MSTR	3495	18CB	• • •
MULHD	**3562**	**1998**	• • •
NEGH	**3524**	**195A**	• • •
NIBL	**33B3**	**17E9**	• • •
NKC	81E5	• • •	• • •
NOCHA	0040	• • •	• • •
NOLIN	0020	• • •	• • •
NOROL	3864	02AD	027E
NOTH	3525	195B	• • •
NROLL	3DA3	0801	07A3
NROLL0	- - -	- - -	07A2
NSEC	000A	• • •	- - -
NTRK	0029	• • •	- - -
NTYP	2D53	1189	• • •
OBC	80E3	• • •	8044
OCODE	80F5	• • •	• • •
ODDFL	81EE	• • •	• • •
OFFPB	26E5	0B88	011D
OPDIR	2D60	1196	• • •
OPEN	**2DAB**	**11E1**	• • •
OPENX	2C86	10BC	• • •
OPENY	2C89	10BF	• • •
OPOX	2C00	1036	• • •
OPX01	2CA2	10DB	10D8
ORAM	80F0	• • •	• • •
ORCHA	3D21	077F	0721
ORHD	352C	1962	1962
OSEC	80ED	• • •	• • •
OSERL	3A61	04B8	0487
OSTR	**33F4**	**182A**	• • •
OSTR1	3404	183A	• • •
OSTR2	340A	1840	• • •
OSTR3	340C	1842	• • •
OSTR4	3410	1846	• • •
OSTR5	3418	184E	• • •
OUTBL	3718	00CD	• • •
OUTCRT	**81C2**	• • •	• • •
OUTFL	**81F8**	• • •	• • •
OUTHL	81FB	• • •	• • •
OVERS	80F6	• • •	• • •
P2SNUM	34D2	1908	• • •
PAGE	3A86	04DD	04AC
PAGE1	- - -	- - -	04AF
PARTMP	- - -	- - -	81B5
PR1	3237	166D	• • •
PBINX	3C05	065C	05FE
PBINX1	- - -	- - -	060A
PBINY	3C8B	06E2	0684
PBINY1	- - -	- - -	0690
PBYT	34CD	1903	• • •
PCFSP	3087	14BD	• • •
PCOLN	34B8	18EE	• • •
PCRAD	81D8	• • •	• • •
PDV	**2FDE**	**1414**	• • •
PDV01	2FF3	1429	• • •
PDV02	3002	1438	• • •
PDV03	300F	1445	• • •
PDV04	3016	144C	• • •
PDV05	3019	144F	• • •
PDV06	3026	145C	145A
PDV07	3055	148B	148D
PDV08	3062	1498	• • •
PFS01	30A5	14DB	• • •
PFS02	30AB	14E1	• • •
PFS03	30BB	14F1	• • •
PFS04	30C1	14F7	• • •
PFSPC	**3077**	**14AD**	• • •
PLINC	3C7C	06D3	0675
PLOFL	81DA	• • •	• • •
PLOKB	3DE9	0847	07E9
PLOTX	3CB2	0709	06AB
PLOTX1	- - -	- - -	06BD

LABEL	v6.78	v8.79	v9.80
PLPTY	3BFD	0654	05F6
PLTAB	3D6F	07CD	076F
PNFSP	**306E**	**14A4**	• • •
PNUM	34D8	190E	• • •
POTON	3D99	07F7	0799
POUND	2511	1E8E	- - -
PPFSP	**3074**	**14AA**	• • •
PRCFPB	- - -	- - -	81AF
PRDEV	2CD4	110A	• • •
PRINT	3A96	04ED	04BC
PRM1	0080	- - -	- - -
PRM2	0081	- - -	- - -
PRM3	0082	- - -	- - -
PRM4	0084	- - -	- - -
PRM5	0085	- - -	- - -
PRM6	0086	- - -	- - -
PRMPT	3382	17B8	17BB
PROCES	398C	03D5	03A4
PROLL	0007	• • •	• • •
PSBYT	34CA	1900	• • •
PSFSP	**3068**	**149E**	• • •
PSNUM	34D5	190B	• • •
PSPAC	34B3	18E9	• • •
PSSTR	34BD	18F3	• • •
PSTAT	81DB	• • •	• • •
PSTR	34C0	18F6	• • •
PTBYT	**324A**	**1680**	• • •
PTREC	**3285**	**16BB**	• • •
PTYP	**2D57**	**118D**	• • •
PUP	81B7	• • •	• • •
PUTEZ	3DE6	0844	07E6
PUTEZ0	- - -	- - -	07E4
PUTXD	3C9B	06F2	0694
PUTXD1	- - -	- - -	069E
PUTXZ	3DC6	0824	07C6
PUTYD	3CEB	0749	06EB
PUTYD0	- - -	- - -	06E5
PUTYD1	- - -	- - -	06F5
PUTYD2	- - -	- - -	06E6
PUTYD3	- - -	- - -	06E8
PUTYZ	3DCD	082B	07CD
PUTZZ	3DAD	080B	07AD
PVREC	**327B**	**16B1**	• • •
PWRUP	37B1	01DF	018E
PWRUP1	- - -	- - -	01DF
PXYCH	3D13	0771	0713
PXYCH0	- - -	- - -	0712
PXZER	3BF3	064A	05EC
Q01	3EBS	0916	• • •
Q02	3EBC	091A	• • •
Q03	3EC9	0927	0929
Q04	3ECB	0929	092B
Q05	3EE7	0945	0947
Q06	3EEF	094D	094F
Q07	3EF7	0955	0957
Q08	3F07	0965	0967
Q09	3F2C	098A	098C
Q10	3F3F	099D	099F

LABEL	v6.78	v8.79	v9.80
Q11	3F43	09A1	09A3
Q12	3F54	09B2	09B4
Q13	3F81	09DF	09E1
Q14	3F85	09E3	09E5
Q15	3F9E	09FC	09FE
Q16	3FAC	0A0A	0A0C
Q20	3FC2	0A20	0A22
Q21	3FCB	0A29	0A2B
QLDA	2AAB	0F4E	0F4B
QTYP	2AAE	0F51	0F4E
RACM	- - -	- - -	00C4
RAMJMP	- - -	- - -	6F80
RATE	- - -	- - -	0002
RATEA	0016	• • •	• • •
RATEB	002F	• • •	• • •
RBLK	**3182**	**15B8**	• • •
RBLKI	31F9	162F	• • •
RBYTE	246A	1DED	- - -
RBYTEC	2474	1DF7	- - -
RCMD	- - -	- - -	0088
RCOMD	0017	• • •	• • •
RD	2EFB	1331	• • •
RD00	2411	1D94	1DC7
RD01	241C	1D9F	1DF3
RD02	2429	1DAC	1DFA
RD03	2435	1DB8	1E04
RD04	2438	1DBB	1E0D
RD05	- - -	- - -	1E11
RDAT	- - -	- - -	0023
REA00	26F1	0B94	• • •
READ	2EA3	12D9	• • •
READY	**81FF**	• • •	**- - -**
REN00	2AC1	0F64	• • •
RERR	2453	1DD6	- - -
RESET	**26A5**	**0B48**	• • •
RF1	2880	0D23	• • •
RF2	2883	0D26	• • •
RFLG	80E1	• • •	- - -
RGAPS	0003	• • •	- - -
RINT	- - -	- - -	0029
RNCOM	- - -	- - -	01A0
RNFIL	- - -	- - -	0ABB
ROLDA	0066	0076	• • •
ROLFL	81DC	• • •	• • •
ROLL	3A85	04DC	04AB
ROLLN	81CD	• • •	• • •
ROT	3EFE	095C	095E
RRTR	000A	• • •	- - -
RS01	26AB	0B4E	0B4B
RS02	26CE	0B71	• • •
RS03	- - -	- - -	0B78
RSCM	- - -	- - -	0004
RSEC	- - -	- - -	0022
RST1J	81C8	• • •	• • •
RSTBF	0002	• • •	• • •
RTST	33D5	180B	• • •
RTST2	33D8	180E	• • •
RUN00	2956	0DF9	• • •

LABEL	v6.78	v8.79	v9.80
RUN01	296D	0E10	• • •
RUN02	297C	0E1F	• • •
RUNC0	- - -	- - -	1A0E
RUNC10	- - -	- - -	1A1D
RUNC20	- - -	- - -	1A29
RWB1	31EC	1622	• • •
RWB2	31EF	1625	• • •
RWBCM	318F	15C5	• • •
RWE	2EC5	12FB	• • •
RWICM	3206	163C	• • •
RWSEQI	**30C6**	**14FC**	• • •
RXBUF	0000	• • •	• • •
RXSER	0020	• • •	• • •
S1OUT	**33C3**	**17F9**	• • •
SAV00	2833	0CD6	• • •
SAVE	3FD0	0A2E	0A30
SBC	8042	• • •	- - -
SBCHA	39BC	0405	03D4
SCMN	3554	198A	• • •
SEC	80EE	• • •	• • •
SEC15	81D7	• • •	• • •
SEEK	2222	1BCE	1C8B
SEEKF	22AA	1C50	- - -
SET00	- - -	- - -	0A76
SETC9	- - -	- - -	0144
SETEXE	- - -	- - -	0A71
SFR	2F96	13CC	• • •
SHIFF	3B85	05DC	057B
SHLHD	353A	1970	• • •
SHRHD	3544	197A	• • •
SIZBIT	- - -	- - -	80E2
SL1	353E	1974	• • •
SLDSD	- - -	- - -	1C50
SOK	0000	• • •	• • •
SP64C	39B4	03FD	03CC
SPNOR	**3460**	**1896**	• • •
SR1	3548	197E	• • •
SRTR	001E	• • •	0004
SSIDA	0008	• • •	• • •
SSOBE	0010	• • •	• • •
STACK	**8042**	• • •	**8044**
START	0000	• • •	• • •
STARTIT	006E	• • •	• • •
STARX	376C	01BA	• • •
STAT5	0003	• • •	• • •
STEP1	- - -	1BD3	- - -
STEPCD	0027	• • •	- - -
STEPS	2525	1E9D	- - -
STIM	003F	• • •	• • •
SPOPIT	006A	• • •	• • •
STOR0	- - -	- - -	0400
STOR1	39EB	0434	0403
STOR2	39F1	043A	0409
STP1	2529	1E9E	- - -
STP2	2536	1EB0	- - -
STP3	2558	1EB7	- - -
STP4	- - -	1ED8	- - -
STPIN	253D	1EBA	- - -

LABEL	v6.78	v8.79	v9.80
STPITN	- - -	1EC0	- - -
STPITO	- - -	1ECE	- - -
STPOUT	254D	1EC8	- - -
STPTIM	0014	• • •	- - -
STPWAT	255E	1EDF	- - -
STRTMP	- - -	- - -	81B3
STWO	2C67	109D	• • •
STX	- - -	- - -	1CAA
STYP	2D5D	1193	• • •
SUBHD	**3459**	**188F**	• • •
SUBU	214D	1AF2	1B97
SULD	2986	0E29	• • •
SVCHA	39DC	0425	03F4
SVCRS	3873	02BC	028D
SVCRS1	- - -	- - -	0290
SVCRS2	- - -	- - -	0293
SXO	2502	1E7F	- - -
SXO1	250C	1E89	- - -
SYSORG	211C	0120	• • •
TAB	38B1	02FA	02C9
TAU	0005	• • •	- - -
TBADDR	368B	- - -	- - -
TBC	80EB	• • •	• • •
TBL00	36A8	005D	• • •
TBL24	36C8	007D	• • •
TBLK	80E7	• • •	• • •
TDRV	80E6	• • •	• • •
TEMP0	81F2	• • •	• • •
TEMP1	81F3	• • •	• • •
TEMP2	81F4	• • •	• • •
TEMP3	81F5	• • •	• • •
TEMP4	81F6	• • •	• • •
TEMP5	81F7	• • •	• • •
TEMPHL	80DE	• • •	• • •
TESHI	39CE	0417	03E6
TEST	3A09	0452	0421
TESTB	3A45	049C	046B
TESTB0	- - -	- - -	0469
TESTB1	- - -	- - -	046C
TESTC	3A2D	0476	0445
TESTC1	- - -	- - -	044B
TESTX	3831	027A	024B
TFCN	80E5	• • •	• • •
TFILE	0002	• • •	• • •
TFREE	0001	• • •	• • •
THRUFL	81DE	• • •	• • •
TIM3X	3EA6	0904	08A6
TIM4X	3AD1	0528	04F7
TIM4X1	- - -	- - -	0501
TIM4Y	3AE0	0537	0506
TIM4Z	3AE6	053D	050C
TIME1	0009	• • •	• • •
TIME2	000A	• • •	• • •
TIME3	000B	• • •	• • •
TIME4	000C	• • •	• • •
TIME5	000D	• • •	• • •
TIMX1	0000	• • •	• • •
TIMX2	0008	• • •	• • •
TIMX3	0018	• • •	• • •
TIMX4	0030	• • •	• • •
TIMX5	0038	• • •	• • •
TMEM	80E9	• • •	• • •
TMODE	39FE	0447	0416
TMP1	81AB	• • •	• • •
TOFF	2154	1AF9	1B9E
TPROG	001B	• • •	• • •
TPROT	0001	• • •	• • •
TRAM	80DE	• • •	• • •
TRK	80EF	• • •	• • •
TWOUT	3B25	057C	0543
TXBUF	0006	• • •	• • •
TXCONT	3B1D	0573	053A
TXOUT	3B30	0587	054E
TXOUT1	- - -	- - -	0550
TXPEN	3B5D	05B4	- - -
TXSER	0028	• • •	• • •
UPDATE	3805	0253	022F
UPTIM	0640	• • •	- - -
VCRAD	81CB	• • •	• • •
VCRSY	3A1F	0468	0437
VE00	2305	1C88	1CF0
VE01	2322	1CA5	1D03
VE02	232A	1CAD	1D09
VE03	2336	1CB9	1D0F
VE04	233D	1CC0	1D1A
VECTO	3E2D	088B	082D
VECTO1	- - -	- - -	082F
VECTY	3BBE	0615	05B4
VERR	230B	1C8E	1CFC
VFILL	**81CE**	• • •	• • •
VHLAD	**81D2**	• • •	• • •
VISIB	3A0A	0453	0422
VRTR	0002	• • •	- - -
VTP	24DD	1E60	- - -
VTP1	24EC	- - -	- - -
VTP2	24F4	1E71	- - -
WATL	**3429**	**185F**	• • •
WATS	**341C**	**1852**	• • •
WB1	245F	1DE2	- - -
WBLK	**317F**	**15B5**	• • •
WBLKI	31F6	162C	• • •
WBYTE	245E	1DE1	- - -
WCMD	- - -	- - -	0024
WCMN	- - -	- - -	00A8
WDAT	- - -	- - -	0027
WDSL	- - -	- - -	0028
WF2	284F	0CF2	• • •
WIG1	22BA	1C53	- - -
WIG2	22C4	- - -	- - -
WIG3	22D2	- - -	- - -
WL1	342A	1860	• • •
WL2	342D	1863	• • •
WR	2EF8	132E	• • •
WR00	23A6	1D29	1D7F
WR01	23BF	1D42	1DA1
WR02	23C4	1D47	1DA7
WR03	23D0	1D53	1DAB
WR04	23D4	1D57	1DB9
WR05	23E6	1D69	1DC4
WR06	23F0	1D73	- - -
WRDIR	2F75	13AB	• • •
WRI00	26EE	0B91	• • •
WRITE	**2ECC**	**1302**	• • •
WRTR	0004	• • •	- - -
WS1	341F	1855	• • •
WSEC	- - -	- - -	0026
WTRK	- - -	- - -	0025
WXYZ	2573	1EF4	- - -
X80	7000	• • •	• • •
XDATA	81EC	• • •	• • •
XFBLK	81A1	• • •	• • •
XFBUF	81A3	• • •	• • •
XFDRV	81A0	• • •	• • •
XFER	219A	1B3F	- - -
XFFCN	819F	• • •	• • •
XFHAN	819D	• • •	• • •
XFXBC	81A5	• • •	• • •
XINTR	0010	• • •	• • •
XORHD	3533	1969	• • •
XOUT0	81AF	• • •	- - -
XOUT1	81B0	• • •	- - -
XTWO	81EA	• • •	• • •
XYMIT	3B13	0569	0530
XYTAB	3D67	07C5	0767
XZERO	81EF	• • •	• • •
YDATA	81ED	• • •	• • •
YTWO	81EB	• • •	• • •
YZERO	81F0	• • •	• • •
ZERFL	39A9	03F2	03C1
ZFATR	8083	• • •	• • •
ZFAUX	809A	• • •	• • •
ZFBLK	80A0	• • •	• • •
ZFBUF	80A2	• • •	• • •
ZFDBK	8098	• • •	• • •
ZFDEN	8099	• • •	• • •
ZFDRV	809F	• • •	• • •
ZFFCN	809E	• • •	• • •
ZFHAN	809C	• • •	• • •
ZFLAD	8093	• • •	• • •
ZFLBC	8092	• • •	• • •
ZFNAM	8084	• • •	• • •
ZFPB	8082	• • •	• • •
ZFPBE	80A8	• • •	• • •
ZFPTR	80A6	• • •	• • •
ZFSAD	8095	• • •	• • •
ZFSBK	808E	• • •	• • •
ZFSIZ	8090	• • •	• • •
ZFTYP	808A	• • •	• • •
ZFVER	808D	• • •	• • •
ZFXBC	80A4	• • •	• • •
ZH	355E	1994	• • •
ZPTR	30D9	150F	• • •
ZRAM	8082	• • •	• • •
ZZZZZZ	3FFA	- - -	- - -

PRODUCT REVIEWS

Joseph Norris

IDA, software debugger
COLORWORD, word processor, screen editor

These two software packages have been long overdue for review by a Compucolor publication. Part of the delay has been caused by revisional activity by the authors, but the real delay has been the lack of adequate marketing efforts for US buyers. Colorcue hopes both these programs will receive the attention they deserve as the latest products directed to the Compucolor/Intecolor owner.

IDA, BY BILL GREENE. $49.50 (INTELLIGENT COMPUTER SYSTEMS)

IDA is a software debugging program by Bill Greene. Bill is also the author of Super Monitor, Super Monitor Plus, Compucalc, and an edition of FORTH, as well a a few games. He has been a frequent contributor to Colorcue since its very beginning.

It's difficult to give a straightforward review of IDA, because after using it daily for several months, I can't think of anything but superlatives. IDA is the topic of the tutorial in this issue by W.S. Whilly, who begins a fascinating exploration of IDA's possibilities. He has only scratched the surface. To be straightforward, it is my opinion that should I lose all my other software, I would hope to keep IDA. It is brilliantly written. It is a flawless performer. It is the greatest timesaver imaginable. IDA is usually the first software I call upon at power-up, and the last I use before dragging my mortal remains to bed. Had I owned it three years ago, I might actually know something about the CCII at this point. It is certain I would have spent much less time head scratching and reconstructing lost disks.

IDA is an (I)nterpreter, (D)isassembler, and (A)ssembler, monitor, calculator and debugger all wrapped within 8K bytes of magic. It is available for loading at 4000H, 8200H, A000H, and E000H. IDA.REL is also available for macroassembler users for loading at any address in user memory. It will operate with v6.78, v8.79 and v9.80 software, and there is work in progress to make it available to 8000 computer users as well.

IDA does all of the following, and then some: a) makes an ASCII dump of memory, b) sets printer Baud rate, c) sets checkpoints, d) disassembles memory, e) prints SRC files, f) fills specified memory with a constant, g) runs assembly programs, h) makes a hex dump of memory, i) interprets assembly programs with a register dump, j) allows addiion of user-defined jump table parameters, k) compares designated memory locations, l) prints any screen display to printer, including displays you type there (in simulated CRT mode) for clarification of screen data, m) moves designated memory, n) makes decimal dump of memory, o) sets origins for pseudo-assembly of mnemonics from keyboard directly to memory, p) peeks, with an option to poke, at any memory location, q) exits to FCS, s) searches memory for byte string, with option to disassemble, replace with designated string, or present a peek/poke mode, t) types keyboard lines to printer, v) displays directory with usual display plus file length in hex and decimal, x) executes any FCS command without exiting IDA, z) sets lines/page and blank line format for printouts.

While IDA has more capability that any previously-issued program of its type, it isn't what it does that is so impressive; it's how it does it! This is difficult to describe in text (you'll do better if you follow W.S. Whilly's article). The large figure on the next page is a "snapshot" of an IDA screen following a disassembly from address 8200. Following the IDA prompt is my instruction to disassemble 10 (hex) program lines beginning at 8200. IDA displays, in order of column, the memory address, a grouping of hex digits associated with that address, a translation of those digits to mnemonics, and, in the last column, an interpretation of the hex digits as though they were not program code.

Notice the lables "IPCRT", "ICRT1", and "KBDFL" in the last column. IDA has recognized these addresses from the disassembly and printed the system names for them —for my convenience. This is a colorful display column, each number being colorcoded to establish its ranking in the 256 possible characters available in the CCII. You can readily identify a printable ASCII character, a control code, or a special character.

Is this particular instance, when the display reached the line of address 8226, I pressed the [UP/ARROW]. A blue bar cursor appeared on the screen. I held the [UP/ARROW] key until the bar cursor moved up to the line it is shown on in the figure, address 820A. IDA now lets me do a number of things at this address: I may use a Peek/Poke option to enter new hex digits at this address, I may set this address as an origin for assembly and actually type in mnemonics at the keyboard, which will be instantly assembled and inserted, beginning at address 820A.

The ease with which this kind of procedure is carried out is not adequately expressed with words. Similarly, all of IDA's functions act with "forethought" which is to say — the program "knows" what you are likely to want next, and presents itself for your service accordingly. It is clearly the work of a master.

The simple feature of making FCS commands available from IDA without a program exit is a remarkable time-saver. No perpetual reentries to IDA, no lost jump vectors from ESC USER. IDA reveals, with ease, any mystery the disk drives hold. All sectors are there to be read with a simple command. All sectors may be rewritten with a simple command.

Calculate with any combination of binary, hex or decimal numbers, using arithmetic operators, or the MOD, AND, OR and EXOR functions. Save yourself from tedious calculations with IDA's display of disk file size. Convert an LDA to a PRG in seconds. Repair bad Basic program lines in a jiffy. Correct a bad disk directory. Do anything you ever wanted to do but couldn't.

Selectively run an assembly language program, one instruction at a time, and

```
FCS>LOAD PNTLOW.PRG
FCS>RUN IDAE

IDA>D8200 10+

8200   210000   LXI    H,0000H    ; ???
8203   39       DAD    SP         ; 9
8204   22E188   SHLD   88E1H      ; "aH
8207   31FF8F   LXI    SP,8FFFH              ; 10
820A   3EC3     MVI    A,C3H      ; >C    195
820C   32C581   STA    81C5H      ; IPCRT
820F   217582   LXI    H,8275H    ; !uB
8212   22C681   SHLD   81C6H      ; ICRT1
8215   3E1F     MVI    A,1FH      ; >_     31
8217   32DF81   STA    81DFH      ; KBDFL
821A   21188C   LXI    H,8C18H    ; !XL
821D   3601     MVI    M,01H      ; 6A      1
821F   AF       XRA    A          ; /
8220   321289   STA    8912H      ; 2RI
8223   321389   STA    8913H      ; 2SI
8226   321489   STA    8914H      ; 2TI
IDA>
```

A SAMPLE IDA SCREEN

watch the registers change...or run sixteen lines at a time, or stop at preselected places of your choosing and examine the registers and memory to see what happened. Look at the stack and watch it change.

All these things are not only routine with IDA, but so logically executed and displayed, it's difficult to remain ignorant about anything for very long.

There must be a catch somewhere! There is. IDA's manual is painfully skimpy. Trying to find out how to make these provisions come to life reminds me of my early days with the Compucolor manual from ICS. The first IDA manual was a skeleton. Subsequent revisions have been improving, but you will be blind to IDA'S power without articles like Mr. Whilly's or a few daring sessions at your own keyboard, exploring.

There are plans at Colorcue to produce a manual for IDA worthy of its content. But, for goodness sake, don't let a poor manual stop you! Get IDA quick, and join us. IDA will be everywhere in the pages of Colorcue. It does for the entire computer what 'THE' Basic Editor and FASBAS have done for Basic. The price is a sinfully modest $49.95. Order from Intelligent Computer Systems in Huntsville, Alabama. (See their catalog, this issue.) If you want a version for 3651 or 8000, write to me at Colorcue. This is a program you'll cherish.

COLORWORD, BY CHRIS TEO. $50. (PROGRAM PACKAGE INSTALLERS)

CCII users have used Comp-U-Writer for years. It's a very respectable word processor at a very high price. One major drawback of Comp-U-Writer is its inability to process control and escape commands to take advantage of the various facilities of modern printers.

COLORWORD removes the price and printer support deficiencies of Comp-U-Writer in one $50 stroke. COLORWORD does not have Comp-U-Writer's "polish", and COLORWORD has its idiosyncracies, but it works, and if you court it appropriately it will serve you well.

The latest revision is v4.5 which contains a new sign-on display (I liked the old one better—less cluttered), keyboard buffering, and a capacity to reformat paragraphs. Current COLORWORD users might want to make the $15 investment to get these improvements if they were out-typing the software and were constantly losing lines of text from the screen.

COLORWORD offers the following, more or less standard, word processing facilities:

Line editing —delete character, delete line, delete all text, insert character, text, page markers and control characters.

Block editing —delete, move, copy, save, and print block.

File commands —save, load, initialize disk, print directory, delete file, rename file, change device.

Cursor control —up down, left right, down/up single or multiple lines, single page.

Printing —Screen preview of printout, print text, set printer parameters (including control and escape codes anywhere in text).

Special functions —string search with optional replace, operate with or without lower case character set (but print either to printer), HELP facility, typematic keys, compact file storage, generates SRC-type files, will process any SRC files, 21K or 27K text area, can be used with any CCII keyboard, available for loading from disk at 4000H.

COLORWORD's manual is thorough and clearly written (18 pages). The only difficulty with this program is learning to live with it's peculiarities. A double RETURN is required to establish a paragraph boundary. When editing (adding) words in the midst of a paragraph you will frequently encounter the "vanishing line" syndrome—entire lines of text disappear from the screen forever. They are in memory, and they print out, but they are invisable. There is a way out of this dilemma, faintly referred to in the manuals of earlier versions, more carefully annotated in v4.5, but one must be on one's toes to keep the demon away. COLORWORD is sometimes erratic in other ways, particularly if you have unusual margins set, or use tabs too much. I haven't found a problem I couldn't circumvent, but I have resented the time I had to spend courting the software when I had pressing work to do. I admit, I have frequently abandoned COLORWORD for Comp-U-Writer. At the same time, the price is right, and it works as advertised. If you've not been able to use a word processor because of a limited budget, COLORWORD is for you. It will be a good friend. COLORWORD works with v6.78, v8.79 and v9.80 computers. It is available from Program Package Installers in Australia. (See their advertisement in this issue.) Don't let their address deter you. PPI is prompt and supports its products conscientiously.

This article has originated from my experiences with the compiler, FASBAS, following the notion that some of the subroutines contained in the Basic interpreter in ROM might be of value in Assembly Language programs. It might be useful to access subroutines for square root, COS, TAN, etc, or the random number generator. The procedure for achieving this is somewhat complicated, but it can be done.

The Basic subroutines all operate on floating point numbers, to handle a wide range of values with accuracy to 5 or 6 decimal places. I assume that Assembly Language programs are going to treat all numbers as integers, expressed as 8 bit or 16 bit binary values. So we must establish a convention, and I propose that any number which is to be operated on by a Basic subroutine must be loaded as a 16 bit value in the DE register pair. Similarly the result of the operation will appear as a 16 bit value in the DE register pair. To accommodate negative values, the most significant bit will be a sign indicator (1 for negative) and thus a range of integers from -32768 to + 32767 can be expressed. This is the normal convention for signed 16 bit binary, in which positive values are counted upward from 0 to 7FFFH, and negative values are counted downward (FFFFH = -1, FFFEH = -2, etc).

So far the strategy is simple and can be described in the following list of steps:

1) Load 16 bit signed value into DE register pair, 2) Call subroutine to convert to floating point, 3) Call required Basic subroutine, and 4) Call subroutine to convert back to 16 bit signed value in DE register pair.

The primary problem is that the value returned by the Basic subroutine will, in many cases, lie between 0 and 1, and will appear as a value of 0 when converted from floating point to binary integer. We can circumvent this by multiplying the floating point number by some scaling factor (such as 100 or 1000) before converting back to integer form, but this is not as straightforward as it seems.

The Basic subroutine for floating point multiplication is not wholly self-contained in ROM. In order to use it we must first call another subroutine which loads instructions into the memory area 8200H to 8298H. Therefore, your program must, at an early point, include (just once) the following instructions (in this and subsequent listings, the Basic subroutine address is given for v6.78, followed by v8.79 in brackets):

CALL 17DEH [372DH] ; Load 8200H to 8298H

Your program cannot itself be loaded at any address lower than 8299H, as it would be overwritten by these Basic instructions.

We need a clear understanding of the order in which Basic carries out a series of mathematical operations, and where it stores intermediate results. Basic uses the memory area 80DEH to 80E1H as a 4 byte accumulator for floating point numbers, and results of all operations are stored here. If the expression COS(X)*100 is to be evaluated, the sequence of operations is as follows:

1. Put the floating point value of X in the accumulator,
2. Call the COS subroutine (result in the accumulator),
3. PUSH the contents of the accumulator onto the stack,
4. Put the floating point value 100 in the accumulator,
5. POP 4 bytes [=COS(X)] from the stack to registers BCDE,
6. Call the multiplication subroutine (result in the accumulator).

This example could be extended to divide the result by Y, in which case Steps 7 and 10 would be very similar to Steps 3 and 6:

7. PUSH the contents of the accumulator onto the stack,
8. Put the floating point value of Y in the accumulator,
9. POP 4 bytes (=COS(X)*100) from the stack to registers BCDE,
10. Call the division subroutine (result in the accumulator).

Two points are worth noting. If X is to be divided by Y, then X must be in registers BCDE and Y must be in the accumulator (and similarly for X-Y). The second point is that all the registers are used during these operations, which may help to explain why Basic moves values from the accumulator to stack and thence to registers BCDE, to keep them safe during Steps 4 and 8. It should also warn you that if you want to retain the contents of any registers, you must PUSH them onto the stack before calling any Basic subroutines.

Using Basic Subroutines in Assembly Language Programming

Peter Hiner 11 Pennycroft Harpenden Herts, AL5 2PD ENGLAND

We are now ready to consider the full sequence for accessing a Basic subroutine from an Assembly Language program. I will use as an example a routine to evaluate the Basic expression INT(RND(1)*100)+1, which generates a random integer value between 1 and 100. I assume that you have previously carried out the instruction CALL 17DEH [372DH] to load Basic routines into the 8200H-8298H memory field. (See Listing 1.)

Note that the subroutine at 134FH [329EH] should always precede CALL 114AH [3099H], even when you are sure the result will be positive.

Listing 2. is a sample of simple subroutines which could be used in the example of Listing 1. where RND was used. For the arithmetic functions, remember that X must be in registers BCDE and Y must be in the accumulator. (See Listing 3. for arithmetic functions.)

A routine to evaluate 3 to the power of 5 is shown in Listing 4. This routine returns a value of 729, so there is no problem, but 3 raised to the 10th power would yield 59049. The hexadecimal value in registers DE would be

Listing 1.

```
LXI   D,1                ; Load 1 for RND(1)
CALL  1A5AH [393ah]      ; Convert 1 to floating point
                         ; - value in accumulator
CALL  16D0H [361FH]      ; RND subroutine generates
                         ; RND(1) in accumulator
CALL  1380H [32CFH]      ; Put RND(1) on stack
LXI   D,100              ; Load scaling factor = 100
CALL  1A5AH [393AH]      ; Convert to fp in accumulator
POP   B                  ; Pop RND(1) from stack to BCDE
POP   D                  ;
CALL  12A3H [31F2H]      ; Multiply RND(1) by 100
CALL  134FH [329EH]      ; Test for positive/negative result
CALL  114AH [3099H]      ; Convert result to integer in DE
MOV   A,E                ; D=0, E=0 to 99
INR   A                  ; Now A=1 to 100
```

Listing 2.

Function..v6.78..v8.79

SQR	15F8H	3547H
RND	16D0H	361FH
LOG	1263H	31B2H
EXP	163FH	358EH
COS	1704H	3653H
SIN	170AH	3659H
TAN	176BH	36BAH
ATN	1780H	36CFH

E6A9H, which, according to our convention for signed 16 bit binary integers, we should evaluate as -6487. Clearly you can, in practice, treat results between 8000H and FFFFH as positive numbers in the range 32768 to 65535, provided that you know what range of result to expect from the Basic subroutines. This does not apply to values at entry to Basic subroutines, which will always evaluate FFFFH as -1, etc. If you try to evaluate an expression which gives a result outside the range -32768 to +65535, then you will find that the program jumps into a Basic error subroutine which gives you a CF ERROR message. Intermediate floating point results outside this range are acceptable provided that you divide by a scaling factor (or add/subtract a suitable value) to bring the result back into range before converting to a binary integer.

A final point to remember is that the conversion from floating point to integer values always gives as a result the lower of the two adjacent integers, and does not round the result to the nearest integer (e.g. 5.9999 becomes 5 rather than 6). You probably expected this, but you may be surprised to learn that -5.001 becomes -6, which is at least consistent if not entirely logical. □

Listing 3.

Function..v6.78..v8.79

X + Y	1167H	30B6H
X - Y	1164H	30B3H
X * Y	12A3H	31F2H
X / Y	12E5H	3234H
X ^ Y	1603H	3552H

Listing 4.

```
LXI   D,3
CALL  1A5AH [393AH]      ; Convert to floating point
CALL  1380H [32CFH]      ; Put 3 on stack
LXI   D,5
CALL  1A5AH [393AH]      ; Convert to floating point
POP   B                  ; POP value 3 to BCDE
POP   D
CALL  1603H [3552H]      ; Subroutine for ^
CALL  134FH [329EH]      ; Test for positive/negative result
CALL  114AH [3099H]      ; Integer in registers DE
```

one dimensional cellular automata

David Suits 49 Karenlee Drive Rochester, NY 14618

By now almost everyone who has worked with computers has learned of the game of *LIFE*, in which the screen is conceptually divided into hundreds of squares (cells), and each cell is treated as containing a living organism (represented, say, by an asterisk) or else treated as dead or empty (represented by a blank)..On the Compucolor/Intecolor it would be appropriate to set up the screen as a two-dimensional array of cells, 64 wide and 32 high. Given an initial number of live cells, placed hither and yon on the screen (usually at the discretion of the user), it is interesting to watch what happens when certain rules of transition are applied to each cell in the array and the new result displayed. In the original *LIFE* there were three rules (or four, depending on how you expressed them) for determining when a live cell "dies", when an empty cell "gives birth" and when live cells stay alive. For example, if a live cell is touching less than two other live cells, the cell dies; when an empty cell has exactly three live cells as neighbors, it gives birth to a new live organism.

A program to run the game of *LIFE* would set up the initial two dimensional array, then check each cell, counting the number of its live neighboring cells and applying the appropriate transition rule. When the entire array has been checked, the program would display the results and then start the transition process over. This would continue 'ad libitum.' It is fascinating to watch cellular patterns emerge and dissolve, especially if the program is implemented in assembly language.

There is nothing sacred, however, about using any particular transition rules which specify when a cell dies or when it stays alive, etc. You can make up your own rules for experimentation. You can also expand upon the concept of a "live" cell: you could have different "species" of orgamisms which interact; each species might have slightly different transition rules.

The game of *LIFE* is an instance of "two dimensional cellular automata." Could there be one dimensional cellular automata? Sure. Instead of n lines of c cells each, you would deal with only 1 line of c cells, and the transition rules would have to deal with a cell's neighbors to the left and right. (A cell

would have no neighboring cells above it, because the line above would represent the previous generation; and it would have no neighboring cells below it, because that would be the next generation which would not yet exist.) In the March, 1984 issue of Scientific American, Brian Hayes discussed one dimensional cellular automata, —a sort of game of *LIFE* played on only one row at a time instead of a full screen. One interesting aspect of this is that each new generation can be displayed on each line of the screen, so that the original line is seen as "growing" from the top of the screen down (or, if you wish, from the bottom up.)

Implementation of a one dimension cellular automaton on the Compucolor or Intecolor computers is easy using plot graphics, where each plot block represents one cell ("alive" if the block is on, "dead" if not). Thus, a one dimensional cellular automaton could be 128 cells long, and you could display 128 lines or generations on your screen. (If you have a printer with dot addressable graphics, you can get even larger automata and print thousands of generations.)

Various transition rules for determining when a live cell dies, when a dead cell gives birth, etc., can be invented. Some transition rules will result in an uninteresting growth (or decay) of cells; others will generate unexpected, plant-like (or crystal-like) structures, often with surprising symmetries.

Listing 1 is a BASIC program which implements a one dimensional cellular automaton which begins at the top of the screen. Each new generation is displayed on the next line down. The transition rule used is simple: for each cell in the line, count the number of live cells two cells to the left and two cells to the right of it, and add 1 if the cell itself is alive. If the total is either 2 or 4, then the cell itself becomes (or remains) alive. Otherwise, it becomes (or remains) dead.

```
0 GOTO 63000 : REM  POKE IN NO-ECHO PATCH
10 REM  ************************************************
11 REM  *                                              *
12 REM  * CELL1D -- One dimensional cellular automata pro-  *
13 REM  *           gram.                              *
14 REM  *                                              *
15 REM  * D. B. Suits,  March, 16 a.l.                 *
16 REM  *                                              *
17 REM  * See Brian Hayes, "Computer Recreations" in    *
18 REM  * SCIENTIFIC AMERICAN, March, 1984              *
19 REM  *                                              *
20 REM  ************************************************
30 REM
40 RGT=127 : REM  Width of line (0--RGT).
50 T0P=127 : BOTT0M=0
60 KB=33278 : KF=33247 : REM  Addresses for keyboard reading.
80 ALIVE=-1 : DEAD= NOT (ALIVE)
90 COL0R=2 : REM  PLOT 6,COL0R for plotting live cells.
100 DIM LINE(RGT),TEMP(RGT)
110 REM
120 REM  ---------------------------------------------
130 REM  Define the transition function for each cell. In the
140 REM  present version, a cell becomes (or stays) alive if
150 REM  the total number of live cells 2 positions on either
160 REM  side of the present cell is either 2 or 4.
170 REM
180 REM  Count live cells in 2 spots left of cell X.
190 DEF FNL2(X)=-(LINE(X-1)=ALIVE)-(LINE(X-2)=ALIVE)
200 REM
210 REM  Count live cells in 2 spots right of cell X.
220 DEF FNR2(X)=-(LINE(X+1)=ALIVE)-(LINE(X+2)=ALIVE)
230 REM
240 GOTO 1000
250 REM
260 REM  This is the actual, called subroutine. It makes use
270 REM  of functions L2 and R2 above if possible. X is the
280 REM  present X index; NC is the Neighbor Count. Boundary
290 REM  checks must be included.
300 REM
310 IF X=0 THEN NC=FNR2(X) : GOTO 360
320 IF X=1 THEN NC=FNR2(X)-(LINE(0)=ALIVE) : GOTO 360
330 IF X=RGT-1 THEN NC=FNL2(X)-(LINE(RGT)=ALIVE) : GOTO 360
340 IF X=RGT THEN NC=FNL2(X) : GOTO 360
350 NC=FNL2(X)+FNR2(X)
360 REM
370 REM  The result is either a live cell or a dead cell,
380 REM  based upon the Neighbor Count.
390 REM
400 IF (NC=2) OR (NC=4) THEN TEMP(X)=ALIVE : GOTO 420
410 TEMP(X)=DEAD
420 RETURN
430 REM
440 REM  ---------------------------------------------
900 REM
910 REM  =============================================
920 REM
930 REM  Main program
940 REM
950 REM  =============================================
960 REM
1000 GOSUB 2000 : REM  Set up and instructions.
1010 GOSUB 3000 : REM  Get initial line.
1015 REM
1020 PLOT 3,64,0,6,COL0R : REM  Hide cursor; set color.
1030 PLOT 2 : REM  Plot mode.
1035 REM
1040 REM  Display initial line at top.
1050 FOR X=0 TO RGT
1055     IF LINE(X)=ALIVE THEN PLOT X,T0P
1060 NEXT
1063 REM
1065 REM  Main loop.
1070 FOR LINE=T0P-1 TO BOTT0M STEP -1
1080     GOSUB 4000 : REM  Calculate and display next line.
1090 NEXT
```

The program is rather slow, so have patience. You can insert some speed-up tricks, but translating the program into assembly language would represent a major speed improvement. (It might be interesting to try a version in FORTH.)

In the BASIC program, the initial line of live/dead cells is entered from the keyboard. The echo is turned off, and a two-high by one-wide plot block "cursor" is drawn. Pressing the space bar enters a "dead" cell and advances the cursor. Pressing any key other than the space bar and RETURN enters a "live" cell and advances the cursor. Pressing RETURN signals the end of input, the rest of the line is filled with "dead" cells, and the automaton begins its slow growth downward. □

COMP—U—WRITER

The popular word processor is still being sold. It was written by a company called INTERSELL for use on the Compucolor and Intecolor series of computers. Several readers have written to Colorcue with questions about the support for this software. We can only give you the names and address of the originators of the program and suggest that you contact them for further information.

INTERSELL
465 Fairchild Drive
Suite 214
Mountain View, CA 94043

Software: Thomas Crispin
Manual: Robert Moody

```
1095 REM
1100 PLOT 255 : REM  Exit plot mode.
1110 INPUT "";A$ : REM  Wait without destroying display.
1120 END
1130 REM
2000 REM  ================================================
2010 REM
2020 REM  Set up and instructions.
2030 REM
2040 PLOT 6,6,29,14,12
2050 PRINT ,,"1-D CELLULAR AUTOMATA"
2060 PRINT
2070 PLOT 15
2080 PRINT "Enter initial line. A space will represent a 'dead'"
2090 PRINT "cell, and any other key (except RETURN) will signal"
2100 PRINT "a 'live' cell. Press RETURN when done."
2110 RETURN
3000 REM
3010 REM  ------------------------------------------------
3020 REM
3030 REM  Get intial line.
3040 REM
3050 PLOT 3,64,0 : REM  Hide cursor.
3060 POKE KF,31 : REM  No-echo.
3070 CX=0 : CY=90 : REM  PLOT X,Y for inputting initial line.
3080 PLOT 30 : REM  Flag on so 2nd PLOT erases previous block.
3090 PLOT 6,COL0R,2
3100 GOSUB 3270 : REM  Display "cursor".
3110 POKE KB,0
3120 A=PEEK(KB) : IF A=0 THEN 3120 : REM  Wait for key.
3130 IF A=13 THEN 3180 : REM  "RETURN" for end of entry.
3140 GOSUB 3270 : REM  PLOT "cursor" a 2nd time to turn it off.
3150 IF A<>ASC(" ") THEN PLOT CX,CY : LINE(CX)=ALIVE : GOTO 3170
3160 LINE(X)=DEAD
3170 CX=CX+1 : IF CX<=127 THEN 3100
3180 POKE KF,12 : REM  Turn on echo.
3190 REM  Fill out initial line if necessary.
3200 IF CX<RGT THEN FOR I=CX TO RGT : LINE(I)=DEAD : NEXT
3210 PLOT 255,29 : REM  Exit plot mode; reset flag.
3220 PLOT 12
3230 RETURN
3240 REM
3250 REM  ------------------------------------------------
3260 REM  Plot the "cursor".
3270 PLOT CX,CY,CX,CY+1 : RETURN
3280 REM
4000 REM  ------------------------------------------------
4010 REM  Calculate next generation.
4020 REM
4030 FOR X=0 TO RGT
4040     GOSUB 310 : REM  Get new generation in TEMP().
4050     IF TEMP(X)=ALIVE THEN PLOT X,LINE
4060 NEXT
4070 REM
4080 REM  Transfer TEMP() to LINE().
4090 FOR X=LFT TO RGT : LINE(X)=TEMP(X) : NEXT
4100 RETURN
4110 REM
4120 REM  ------------------------------------------------
62000 REM
62010 REM  Ben Barlow's No-echo patch.
62020 REM
63000 RESTORE 63000 : DATA 245,175,50,255,129,241,201
63010 TM=256*PEEK(32941)+PEEK(32940)-7
63020 FOR X=1 TO 7 : READ D : POKE TM+X,D : NEXT
63030 BR=INT(TM/256) : POKE 33221,195 : POKE 33222,TM-BR*256+1
63040 POKE 33223,BR : POKE 32941,BR : POKE 32940,TM-BR*256
63050 CLEAR 50 : GOTO 10
```

KVHGRxRWRO KILTIZnNrmT

"...NLERMT RMGL GSY FKKVI IGMPH."

DQRIIB HIN DSROOB

I am aware of the patronizing attitude taken by some individuals, who learn something of value, perhaps print a few silly articles, and then retire with their wisdom forever. They reach some kind of summit from which they can no longer share, and decide all further questions are stupid. Have you, for example, ever seen an article that tells how to perform routine system debugging? Now you know why I have emerged from obscurity.

The idea of "bugs" in a computer system derives, so legend has it, from early experiences with vacuum tube computers at the Department of Defense, which were known to fail because of nests of moths and other little creatures. From this, we speak of our failures as being the responsibility of "bugs" in almost anything. So a "debugger" is meant to remove the human errors from software and hardware systems. We are discussing software debuggers.

There are five software debugging tools currently available for the CCII and 3651 computers; MLDP by ISC, DEBUG and NEWBUG by Com-tronics, Super Monitor Plus and IDA by Bill Greene. These are listed in increasing order of versatility, in my opinion, although each has its own special, useful features. MLDP differs from the others by its lack of printing facilities and is frustrating in this regard for serious programming. MLDP does, however, have a unique monitor which we shall examine next time. Most of the others are somewhat similar in their capabilities, but IDA stands out far above the rest. If you don't have IDA you are missing the companionship of a true friend, for it is the best there is, and an invaluable working and teaching aid.

These articles will tend to focus on IDA , but if you have one of the others and don't want to purchase IDA, I will include for you those commands which are somewhat comparable in MLDP and DEBUG, as far as they go. Super Monitor Plus is a "subset" of IDA, so many of IDA's commands will work with it as well. In addition to one or more of these debugging programs, you will need an assembled version of CYPHER (at 8200H) from the MAR/APR Colorcue, on a disk by itself, and another disk side with no valuable material on it. Any directory on this second disk will be obliterated in the course of this article. Your work will be facilitated by a printout of the assembly of CYPHER (see Colorcue Mar/Apr 84), showing addresses and mnemonic code.

A "debugging" software tool is a program that allows you to manipulate the contents of an FCS file, whether it be a PRG, BAS, SRC, LDA, or any other kind of file. The debugger is most often used with PRG files, to "run" them in selected portions and to examine what is taking place as they run. This kind of activity exercises the "monitoring" capability of a debugger. The debugger permits changing program code, in a limited fashion, without the necessity for reassembly, and it permits disassembling PRG files into their source code. A good debugger will permit you to correct faulty BASIC code lines that are making a program unexecutable. You may print a file in hex number form, or ASCII form. You may move code from one set of addresses to another. The list of possible uses is truly endless. I will introduce you to the power of debugging in a logical and disciplined way, using the program CYPHER.PRG.

A debugger is usually loaded into high memory, above the area used by most FCS files. This permits two programs to exist simultaneously in memory. In these articles the debugger is called the "instrument" and the program being worked on is called the "subject." If the subject code occupies some of the memory required by the instrument then debugging in the normal way is made impossible. If the subject only uses instrument memory to store data files, then debugging is still sometimes possible, even with this conflict. (IDA is available for loading at 4000H which keeps it out of the way, and therefore usable with any program that will fit into 32K CCII memory.)

To set up the instrument, the usual procedure is to load or run the subject program first from FCS. The LOAD instruction defaults to LDA, so if your program has a different file type, the load instruction must contain the file extension; example: LOAD CYPHER.PRG. With this command, CYPHER will be loaded into memory but will not "run." If the RUN command is used, the default type becomes PRG, so the command will read: RUN CYPHER. In this case, CYPHER will load and run both. To stop program execution, press [CPU/RESET] to exit, and [ESC] [D] to return to the file control system.

Debuggers will generally load at several of the addresses 4000H, 8200H, A000H, and E000H (look at the available loading addresses of your instrument on its disk directory, then pick a version that will load safely out of the way of your subject program). The last letter of the IDA name is the first letter of the loading address. (Some IDA versions are just named "8.PRG", "E.PRG", etc.) IDAE loads and runs at E000H. IDA8 loads and runs at 8200H, and so forth. If you have 16K memory and a subject at 8200H, then you must satisfy yourself with MLDP16 or DBUG16, or IDAA at A000H. If the subject program loads at E000H, then your debugger will have to be loaded below that, say at 8200H, so it does not conflict with the subject. To invoke the debugger, type from FCS a run command with the debugger name; example: RUN IDAE, or RUN DBUG32, or whatever.

Now it's time to get your feet wet. "LOAD" CYPHER.PRG, and when the FCS prompt reappears, RUN IDAE or some other instrument (into high memory). CYPHER begins at 8200H and ends at 847DH, so any of A000H, E000H or 4000H will do for the instrument. Since

we're loading into high memory, version ;02 of MLDP and DEBUG will be used, so a version need not be specified (unless you have ;03 for some reason):

```
[Note: Later versions of IDA use [ESC] for exits from all
IDA functions.   [ESC] will replace [DOWN/ARROW] where it
is specified in these articles.]
```

```
IDA (SUPER MONITOR):

FCS>RUN CYPHER
[CPU/RESET] [ESC] [D]
FCS>RUN IDAE
IDA>
```

```
DBUG32:

FCS>RUN CYPHER
[CPU/RESET] [ESC] [D]
FCS>RUN DBUG32
COMMAND>

MLDP:

FCS>RUN CYPHER
[CPU/RESET] [ESC] [D]
FCS>RUN MLDP32
DBG>
```

The last lines above illustrate the instruments' prompts, ready to receive a command. It is expected that when a command line is keyed in, you will follow with a [RET]. On DEBUG there are some commands that will result in an automatic RETURN. DEBUG also performs some automatic "space-overs" on command lines with multiple parameters. It will take a little "getting used to." We may now begin work. Let us look first at the elementary process of disassembly, in which the machine code is translated into the more readable mnemonics of the source code from which it was derived. The display will differ from original source code in that all labels will be replaced by their actual assembled addresses. The disassembler feature generally requires a disassembly command, a starting address, and a length of addresses to be disassembled. This length may take the form of an ending address or a number of addresses following the starting address. We will look at the first few complete code lines of CYPHER on the CRT.

```
IDA>D 8200 10+                    COMMAND>S 8200 821F
[DOWN/ARROW] or [ESC]

                                  DBG>DS 8200,20
```

What is seen with disassembly varies among the instruments. IDA will give the most spectacular readout (see LIST A), however, in this disassembly mode, most programs are very much alike. In List A, column 1 displays the memory addresses, column 2 shows the hexadecimal bytes located in those addresses (clustered in IDA, separated in other instruments), columns 3 and 4 show the disassembled mnemonic form of the code (columns 4 and 5 for MLDP), and column 5 prints a translation of column 2 which interprets memory as an array of characters and not mnemonic code. (In MLDP, this display appears as column 3, which "strips" lower case to upper case. The only way to tell if you're looking at a lower case character is to examine the hex code.) Look at address 820CH. The first byte represents the op-code for "CALL" (CD). The next two bytes are the low and high bytes of the address of the CALL (2A, 18 for v8.79 and v9.80, F4, 33 for v6.78). This line is calling OSTR in ROM. If you have IDA, in column 5 you will see "OSTR" printed. IDA translates all frequently used ROM addresses into their source code name (very helpful if you are modifying a PRG file for another version of system software). To find what OSTR is about to print, look at address 8209H and you will find the HL registers loaded with address 82D7H, which just happens to begin the printable string—6,2,3,0,10,'NORMAL TEXT'—etc. All the lines in Listing A are instructional lines of code, including the three NOPs at the beginning which give the instruction to do

Now let's move into the "high weeds" (as Myron used to say). Repeat the last disassembly but start at 8204H. You should get something like List B, and it looks nothing like List A at all until it reaches address 8206H, even though it has all but one byte of List A (not counting NOPs) in its disassembly. How come? The disassembler began to decode the memory contents at address 8204H, where it saw 25H, which happens to be the op-code for"DCR H." The disassembler will try to interpret the first byte it sees as an op-code. That isn't what we intend the 25H to mean. We mean it to be the low byte of the address of string CLR (at

```
IDA>D 8200 15+                    List A

8200   00       NOP                    ; @
8201   00       NOP                    ; @
8202   00       NOP                    ; @
8203   212583   LXI    H,8325H         ; !%C
8206   CD2A18   CALL   182AH          ; OSTR
8209   21D782   LXI    H,82D7H         ; !WB
820C   CD2A18   CALL   182AH          ; OSTR
820F   212083   LXI    H,8320H         ; ! C
8212   3600     MVI    M,00H           ; 6@     0
8214   23       INX    H               ; #
8215   360C     MVI    M,0CH           ; 6L    12
8217   0E00     MVI    C,00H           ; N@     0
8219   212383   LXI    H,8323H         ; !#C
821C   3602     MVI    M,02H           ; 6B     2
821E   3EC3     MVI    A,C3H           ; >C   195
8220   32C581   STA    81C5H           ; IPCRT
8223   21A782   LXI    H,82A7H         ; !'B
8226   22C681   SHLD   81C6H           ; ICRT1
8229   3E1F     MVI    A,1FH           ; >_    31
822B   32DF81   STA    81DFH           ; KBDFL
822E   C35E82   JMP    825EH           ; C^B
```

8325H). If the computer is going to see it that way, we **must** begin disassembly in such a way that the computer will recognize 25H as a low byte address of a preceeding op-code, specifically here, 21H, the op-code for "LXI, H", which is at address 8203. To satisfy this requirement, we must always begin disassembly at some single byte instruction or single byte of independent data, such as a DB byte. Otherwise, we'll be "in the high weeds" and working against the wind. If you're in a program unknown to you, then you must "feel" your way from the starting address, writing down legitimate starting places as you disassemble your way through the program. If this isn't clear, try disassembling at each address in turn, from 8200H to 820BH and see when the disassembly is congruent to List A and when it isn't.

The procedure for controlling the number of disassembly lines varies among instruments. In our examples, IDA is given the number of lines to decode (in hex), DEBUG is given the last line to decode, and MLDP is given the number of bytes to decode (hex). If the end specification falls at the beginning or in the midst of a two or three-byte instruction, the entire instruction will be decoded.

Now, using the same technique used to generate List A, disassemble the code beginning at 8340H. IDA will accept a number of different disassembly specifications, but I like specifying the number of lines the best. DEBUG wants the start and end address. MLDP wants the number of bytes to decode:

```
IDA>D 8340 10+      COMMAND>S 8340 8351      DBG>DS 8340,12
```

This will look very much the same for all the instruments, and is shown in List C. What you see in the mnemonic columns is interesting but only garbage. The "real" meaning

```
IDA>D 8204 15+                List B

8204   25       DCR    H          ; %
8205   83       ADD    E          ; C
8206   CD2A18   CALL   182AH       ; OSTR
8209   21D782   LXI    H,82D7H    ; !WB
820C   CD2A18   CALL   182AH       ; OSTR
820F   212083   LXI    H,8320H    ; ! C
8212   3600     MVI    M,00H      ; 6@     0
8214   23       INX    H          ; #
8215   360C     MVI    M,0CH      ; 6L    12
8217   0E00     MVI    C,00H      ; N@     0
8219   212383   LXI    H,8323H    ; !#C
821C   3602     MVI    M,02H      ; 6B     2
821E   3EC3     MVI    A,C3H      ; >C   195
8220   32C581   STA    81C5H      ; IPCRT
8223   21A782   LXI    H,82A7H    ; !'B
8226   22C681   SHLD   81C6H      ; ICRT1
8229   3E1F     MVI    A,1FH      ; >_    31
822B   32DF81   STA    81DFH      ; KBDFL
822E   C35E82   JMP    825EH      ; C^B
```

of these memory contents is in column 5 (column 3 for MLDP)—a translation of a string of ASCII characters. How do we know? For one thing, the disassembled mnemonics don't make sense. For another, the ASCII translation does. Furthermore, using the hard copy of the assembled version of CYPHER, we can trace this location back to address 8203, where the LXI H,CLR (8325H) instruction appears. OSTR will begin printing at 8325H and continue until it reaches a "239". (We are viewing a mid-section of this long string.) The first subsequent "239" occurs at address 8429H (EF). This won't just come to you, you have to study the printout of CYPHER to see it.

The subsequent area of memory contains the misspelled words, and we are going to correct them without reassembling CYPHER. Instruments allow you to change memory contents. Disassemble from 8352H and observe the word "prmgram". We will first replace the bad "m" in "prmgram" with a good "o". (If you don't have lower case, just use capitals.)

```
IDA>D 8352 10+                COMMAND>S 8352 8362
Press [UP/ARROW]              COMMAND>E 835B
 (Bar cursor appears)         835B: 6D-
Hold [UP/ARROW]               (Enter 6F)
 until it is on "m".          [RET]
Press [P] (peek/poke)         835C: 67-
 ASCII representation         [DOWN/ARROW]
 will appear on bottom        COMMAND>
 of screen. Cursor is
 under the "m."
Now press [INSERT/CHAR]                    DBG>DS 8352,12
 and lower case "o".                       DBG>@835B
 Code will appear in          835B 6D 'M' MOV L,L
 hex, but ASCII will          MEM>=6F
 not change.                  835C 67 'G' MOV H,A
Press [RET] to exit the       MEM>/
 Peek/Poke mode.              DBG>
```

To see the wonderous results of your work, disassemble again from 8352H. We can get more practice at this by correcting misspelling at addresses 83B0 and 83DD. So do that now on your own.

Reference: "i" (69H), "I" (49H), "a" (61H), "A" (41H)

Instruments allow you to "run" a program in memory. Now "run" the program and observe the corrected spelling.

```
IDA>G 8200          COMMAND>G 8200          DBG>R 8200
```

After admiring your new spelling, note that the first "a" of "according" is at the end of a line, with the rest of the word on the following line. We can correct that easily with IDA.

```
IDA>D 8340 15+                List C

8340   54       MOV    D,H        ; T
8341   6F       MOV    L,A        ; o
8342   20       - -               ;
8343   45       MOV    B,L        ; E
8344   6E       MOV    L,M        ; n
8345   63       MOV    H,E        ; c
8346   6F       MOV    L,A        ; o
8347   64       MOV    H,H        ; d
8348   65       MOV    H,L        ; e
8349   20       - -               ;
834A   54       MOV    D,H        ; T
834B   65       MOV    H,L        ; e
834C   78       MOV    A,B        ; x
834D   74       MOV    M,H        ; t
834E   2E03     MVI    L,03H      ; .C     3
8350   00       NOP               ; @
8351   02       STAX   B          ; B
8352   0602     MVI    B,02H      ; FB     2
8354   54       MOV    D,H        ; T
8355   68       MOV    L,B        ; h
8356   69       MOV    L,C        ; i
```

Press [CPU/RESET] to exit CYPHER. Reenter any of the instruments by pressing [ESC] [user]. Now disassemble beginning at 8390 (List D). If we could place a carriage return and line feed after "it", then "according" would be altogether on the next line. The space (20H) between "it" and "according" can be used for the carriage return (we don't need the space at the end of the line anymore), but we need another memory location for the line feed. If we read the entire string given to OSTR, beginning at address 82D7, we find that there is unused memory following the "239" at address 842AH. Therefore, if we moved all the code, extending from 8393 to 8429, down one address, we would free a memory location at 8393 for the line feed.

```
IDA>D 8390 10+  List D (Before changes)

8390   69      MOV     L,C         ; i
8391   74      MOV     M,H         ; t
8392   20      - -                 ;
8393   61      MOV     H,C         ; a
8394   63      MOV     H,E         ; c
8395   63      MOV     H,E         ; c
8396   6F      MOV     L,A         ; o
8397   72      MOV     M,D         ; r
8398   64      MOV     H,H         ; d
8399   69      MOV     L,C         ; i
839A   6E      MOV     L,M         ; n
839B   67      MOV     H,A         ; g
839C   20      - -                 ;
839D   74      MOV     M,H         ; t
839E   6F      MOV     L,A         ; o
839F   20      - -                 ;
```

Instruments allow you to move blocks of memory. The MOVE instruction has a starting and ending address of the block to be moved, and the new starting address of the block after the move. The "move" is really a "copy", since the original contents will not be altered unless it is written over. IDA will move from anywhere to anywhere, but DEBUG and MLDP will not. For these last two, you must move the memory area completely outside the working area in which it currently resides, and then move it back in again with a one-byte displacement. If you have DEBUG or MLDP, fill the area from A000 to B000 with 00 (to unclutter it and make examination easier). Move the string into that area, then move it back where we want it ..and think about saving money to purchase IDA. (My MLDP manual has an error for MOVE instructions. It is shown correctly here):

```
IDA>M 8393 8429 8394        COMMAND>Z A000 B000 00
                            COMMAND>M 8393 842A A000
                            COMMAND>M A000 A097 8394

                                        DBG>F A000:B000:00
                                        DBG>M 8393:842A TO A000
                                        DBG>M A000:A097 TO 8394
```

(Move the range 8292-8429 to the range beginning at 8394.)

Disassemble from 8390 and notice two "a"s at 8393 and 8394. The first "a" can be replaced by a line feed.

```
IDA>D 8390 10+              COMMAND>E 8392          DBG>@8392
[UP/ARROW] to 8392          8392: 20-               8392 20 ' '
Press [P]                   (Enter 0D)              MEM)=0D
Enter 0D (at 8392)          8393: 61-               8393 61 'A'
Enter 0A (at 8393)          (Enter 0A)              MEM)=0A
[RET] to exit               [DOWN/ARROW]            8394 61 'A'
                                                    MEM)/
```

Now check it out. (See 'after' List E.) Run the program with 'G 8200' (or 'R 8200' for MLDP.)

Isn't that better! IDA is unique in that it can move bytes around in a very tight space. Try this with DEBUG or MLDP and you'll have an interesting experience.

We don't want to have to do these corrections twice, so let's save a new CYPHER on disk. Note we haven't increased the length of CYPHER, only changed its contents. You may use FCS commands to save a "memory image" (a picture of memory) on disk. This memory image just happens to be a working PRG program. Most instruments will make this SAVE from FCS, but if you have IDA, you can return to the instrument and do it from there.

```
IDA>XSAVE CYPHER.PRG 8200-847D

          MLDP AND DBUG: FCS>SAVE CYPHER.PRG 8200-847D
                         FCS>[ESC] [^]
```

We will now erase all the memory used by CYPHER plus a little more. Instruments allow you to fill a range of memory with any byte you may choose. The FILL instruction usually has a starting address, an ending address, and the byte with which you wish to fill:

```
IDA>F 8200 8500 00          COMMAND>Z 8200 8500 00

                                        DBG>F8200:8500:00
```

(Fill the memory range 8200-8500 with byte "00.") Verify the fill by disassembling from 8200H. All NOPs.

```
IDA>D 8390 10+  List E (After changes)

8390   69      MOV     L,C         ; i
8391   74      MOV     M,H         ; t
8392   0D      DCR     C           ; M  < CTRL M = cr
8393   0A      LDAX    B           ; J  < CTRL J = lf
8394   61      MOV     H,C         ; a
8395   63      MOV     H,E         ; c
8396   63      MOV     H,E         ; c
8397   6F      MOV     L,A         ; o
8398   72      MOV     M,D         ; r
8399   64      MOV     H,H         ; d
839A   69      MOV     L,C         ; i
839B   6E      MOV     L,M         ; n
839C   67      MOV     H,A         ; g
839D   20      - -                 ;
839E   74      MOV     M,H         ; t
839F   6F      MOV     L,A         ; o
```

There is another way to load CYPHER into memory for further work. We know it isn't there now. Call up your disk directory and write down the starting block (SBLK) of the newest CYPHER (the corrected version.) My starting block is 207. (I have an 8'' drive. See List H.) You will use your own appropriate block number where I have written ''207'' in the instructions:

```
IDA>XDIR

                DEBUG AND MLDP: [CPU/RESET] [ESC] [D]
                                FCS>DIR
```

We will use the FCS ''REAd'' command to get CYPHER this time. Most instruments will require that you do this from FCS, but IDA can do it from the IDA prompt. The REAd command contains the starting disk block number, the start address in memory and the end address in memory where the code will be placed. We can get the start and end addresses from the assembler printout of CYPHER:

```
IDA>XREA 207 8200-847D

                DEBUG AND MLDP:  FCS>REA 207 8200-847D
                                 FCS>[ESC] [^]
```

(Read from disk beginning at block 207, into memory beginning at 8200, until memory is filled to 847D.)

This loads CYPHER directly. Notice how fast this load is. Speed is increased because the operating system doesn't have to work with the directory. Disassemble from 8200H to verify this load. Now we are going to write a new subroutine into memory and use it. Notice the instruction, at 8206H, CALL 182A (CALL OSTR). (Your address will be 33F4 for v6.78.) This is the famous output string subroutine in ROM. Let's not use it from ROM, but write our own version of OSTR inside CYPHER. When OSTR is called, the HL registers hold the address of the first byte to be printed. The last byte is followed by ''239.'' This routine (which we shall call OMSG) will do:

```
Label Mnemonics Operation..................... Address  Hex Code

OMSG: MOV A,M ; Move memory byte to accummulator  842D      7E
      CPI 239 ; Is it the end? (239=EFH)          842E      FE EF
      RZ      ; Yes, return to caller             8430      C8
      CALL LO ; No, print this byte               8431*     CD C8 17
      INX H   ; Point to next memory location     8434      23
      JMP OMSG;  and get next byte                8435      C3 2D 84

                    * "CD F4 33" for v6.78.
```

There is room for this routine at 842D (leaving a few bytes free beyond the ASCII string). We cannot use it as shown above. for we must replace all the labels with actual addresses. Instruments allow you to insert op codes and their accompanying addresses in memory in the form of hex numbers. IDA will let you enter the mnemonics directly, the rest of you will have to poke in the hex numbers, one at a time from the table above:

```
IDA>D 842A 15+
[UP/ARROW] to 842D
Press [O] to set
 origin for IDA's
 assembler.
ASM>MOV A,M
ASM>CPI 239
ASM>RZ
ASM>CALL 17C8 (*)
ASM>INX H
ASM>JMP 842D
[RET]

* Use 33F4 for v6.78 (F4, 33)
```

```
COMMAND>E 842D
842D: FF-
(Enter  7E
        FE
        EF
        C8
        CD
        C8 (*)
        17 (*)
        23
        C3
        2D
        84
[DOWN/ARROW]
```

```
DBG>@842D
842D FF '_' RST 7
MEM>=7E
MEM>=FE
MEM>=EF
MEM>=C8
MEM>=CD
MEM>=C8 (*)
MEM>=17 (*)
MEM>=23
MEM>=C3
MEM>=2D
MEM>=84
MEM>/
```

Disassemble from 842A to verify that the subroutine has been entered (List F). Now we have to call 842D every time we formerly called 182A. IDA makes it easy to find all the references to OSTR in our program. We will conduct a ''simple search'' for the code string ''CD 2A 18'' (or CD F4 33), placing the low byte first for the OSTR address. While these address bytes ''read right'' in the disassembly, you know by now that they are reversed in memory:

```
IDA>SS 8200 8500 CD 2A 18
```

```
COMMAND>F 8200 847D CD 28 1A  (Sorry. MLDP can't
                               do this.)
```

IDA and DEBUG locate this code at four places: 8206, 820C, 827D, 8290. With MLDP, you would have to disassemble the entire file, and watch for OSTR's address, making note of its occurrances. IDA will also replace the code at these addresses if we ask it to. We perform a ''search and replace.'' When using this function IDA asks for your ''replace'' string (RPL):

```
IDA>SR 8200 8500 CD 2A 18
RPL>CD 2D 84

While they're doing their
thing (to the right),
let's check it out by
disassembling at least
one address, using
IDA>G 8200, to watch our
new OMSG at work! By that
time maybe the others will
be ready to go on!
[CPU/RESET] [ESC] [^] to
return to IDA>.
```

```
COMMAND>E 8206
8206: CD-
[RET]
8207: 2A-
(Enter 2D
8208: 18-
(Enter 84
[DOWN/ARROW]
COMMAND>E 827D
827D: CD-
[RET]
827E: 2A-
(Enter 2D
(etc.)
```

```
DBG>@8206
8206 CD 2A 18
MEM>=CD
MEM>=2D
MEM>=84
MEM>=/
DBG>@827D
MEM>=CD
MEM>=2D
MEM>=84
MEM>/
(etc.)
```

(Be sure you change all four addresses.)

We can now save this new version of CYPHER, but rather than create still another new disk file, we will write new version 2 over new version 1. We can do this easily with the FCS WRIte instruction. Using the same SBLK as before we will transfer new version 2 to disk. Substitute your own disk block number for my "207" again:

```
IDA>XWRI 207 8200-847D

                DEBUG AND MLDP: [CPU/RESET] [ESC] [D]
                                FCS>WRI 207 8200-847D
```

Now let's write another copy of CYPHER to a clean disk. Remove the first disk and put a clean disk in the drive. We don't really need a directory, you know, so let's write CYPHER beginning at block 00. Do not use a disk for this procedure that contains anything you want to keep. Any directory on this disk will be destroyed:

```
IDA>XWRI 00 8200-847D

                DEBUG AND MLDP: FCS>WRI 207 8200-847D
```

Now look (or try to look) at this disk directory. You will get an ENVE error because there is no directory. But CYPHER is there, my friends. Fill 8200-8500 with 00 again and verify that this memory is cleared. Now read CYPHER back into memory from the same no-longer-clean disk:

```
IDA>XREA 00 8200-847D

                DEBUG AND MLDP: FCS>REA 00 8200-847D
                                FCS>[ESC] [^]
```

Disassemble from 8200H. There it is! Isn't this great fun! You can "run" the program with "G 8200" if you are skeptical.

For our last act this time, we are going to change the encoding used by CYPHER. We will be original and let "A" = "A", "B" = "B", and so on. This is an easy change to make. The encoding table begins at 8448. The first letter ("A") is the "normal" letter of your keyboard input. It is followed by the letter to which the first letter will be encoded ("Z"). This correspondence between pairs of letters continues to 847E. The last two entries, at 847C and 847D "convert" spaces to spaces, to allow them as legal entries into the code. (Any character not represented in the table is not recognized. Does that give you any ideas for the space beginning at 847E?) List G is IDA's printout of this memory range in ASCII. The other instruments will make a hex dump of this table to use for reference as you poke in the new data. Be careful you get the right address for data entry:

```
IDA>D 8448 10+
[UP/ARROW] to 8448.


Press [P] for Peek/Poke
option.
[RIGHT/ARROW] to 8449
Press [INSERT/CHAR] [A]
[RIGHT/ARROW] to 844B
Press [INSERT/CHAR] [B]
Continue until all
 letters are changed.
At the end of the first
 line of ASCII display,
 the next line will be
 displayed automatically.
Press [RET] to return
 to IDA>.
```

```
COMMAND>H 8448 847E


COMMAND>E 8449
8449: 5A-
(Enter 41
[RET] [RET]
(Enter 42
[RET] [RET]
(Enter 43
(etc.)
.........
(Enter 5A
[RET]
[DOWN/ARROW]

DBG>D 8448:847E
(Error in MLDP manual
for this command. It
is shown correctly here.
DBG>@8449
8449 5A 'Z'
MEM>=41
[RET] [RET]
MEM>=42
[RET] [RET]
MEM>=43
(etc.)
..........
MEM>=5A
847C: 20
MEM>/
```

Verify the new encoding table with a dump from 8448 to 847D. You should see pairs of hex numbers from 41 to 5A. You can also try the program by running it again.

We are going to save this new encoding table as a separate disk file. We will put it on the disk with no directory. In order to do this, we need to know how much disk space CYPHER occupies, so we won't overwrite CYPHER accidently. CYPHER begins at 8200 and ends at 847D. Let's calculate the number of bytes. (There is a mistake in my MLDP manual regarding calculations. The "/" symbol is not to be used at all. Numbers entered directly will be assumed as hexadecimal numbers. Numbers preceeded by a "#" will be taken as decimal.)

```
IDA>847D-8200=                          DBG>=847D-8200
827D  637  00000010 01111101            =827D #637
```

Calculations say the program is 027DH or 637 decimal bytes long, but we must also add the end byte, which makes 638. Wait a minute! With IDA we don't have to calculate this. If we enter a "V" at the IDA prompt, IDA prints the usual directory but adds the file size in hex and decimal for us (List H). Well, anyway, how many blocks is this (at 128 bytes/block)?

```
IDA>027D/%128=                          DBG>027D/#128
```

The calculators say 4 blocks, but they are performing integer arithmetic. It's actually 4.98 blocks long. We'll say 5 blocks because we must write with integer values of block numbers. FCS begins numbering its blocks with 00, so the sixth block, where we might plan to put our new table, is actually referred to as block "5". For some reasons that might become apparent later on, let's store it beginning at block 0A. The code table begins at 8448 and ends at 847B:

```
IDA>XWRI 0A 8448-847D

                DEBUG AND MLDP:  [CPU/RESET] [ESC] [D]
                                 FCS>WRI 0A 8448-847D
```

Now we can use either code table. If we REAd CYPHER to memory from block 00 we will have the original table:

```
IDA>XREA 00 8200-847D

                DEBUG AND MLDP:  FCS>REA 00 8200-847D
```

If we "overlay" the code table from block 0A we will have the amended code table:

```
IDA>XREA 0A 8448-847D
```

There is no directory to give tell-tale clues about the disk contents. We have a rather well-protected encryption procedure as long as we don't forget which block is which.

Well, let's try one more act. Load CYPHER.PRG back into memory from block 00. Now initialize the "clean" disk we have just been using, but only allow "five" directory blocks:

```
IDA>XINI CD0:FREE 5

                DEBUG AND MLDP:  FCS>INI CD0:FREE 5
```

Call up the directory and find the first available unused block (SBLK for "Free Space.") WRIte CYPHER to that start block (my SBLK is 05):

```
IDA>XWRI 05 8200-847D

                DEBUG AND MLDP:  FCS>WRI 05 8200-847D
```

If you look at the directory again, you'll notice CYPHER isn't listed even though we just "wrote" it there. Now go into BASIC by pressing in sequence [CPU/RESET] [ESC] [E]. We are going to create a directory entry for CYPHER.PRG from BASIC! Since CYPHER takes nearly five blocks, we can reserve 5 * 128 bytes/block for it (=640 bytes total). Type the following in "immediate" mode:

```
FILE "N", CYPHER.PRG, 1,640,1
```

Call up the directory from FCS. We have a record of CYPHER's presence. Type "RUN CYPHER." Honestly now, did you really think it was going to work? Well, it can if we put in the correct loading and starting addresses..... next time. Meanwhile, you can WRIte it into memory and "run" it from your instrument; no one else is going to RUN your secret encoder! If you wrote CYPHER to disk beginning at block 05, then the "overlay" is still intact at block 0A. You can load it just as we did before. For homework, why not create a directory entry for the overlay? More fun to come!□ *W. S. Whilly*

Note: All listings and their comments were printed directly from IDA's screen. A "CRT mode" facility permits writing comments anywhere on the screen for inclusion in printouts. Virtually any IDA screen display may be sent to the printer during the course of work.

```
IDA>D 842A 15+          List F  (DMSG)

842A   EF       RST     5           ; o
842B   FF       RST     7           ;
842C   FF       RST     7           ;
842D   7E       MOV     A,M         ; ~          (
842E   FEEF     CPI     EFH         ; ~o   239   (
8430   C8       RZ                  ; H          (
8431   CDC817   CALL    17C8H       ; LO         (
8434   23       INX     H           ; #          (
8435   C32D84   JMP     842DH       ; C-D        (

8438   FF       RST     7           ;
8439   FF       RST     7           ;
843A   FF       RST     7           ;
843B   FF       RST     7           ;
843C   FF       RST     7           ;
```

```
IIDA>P 8448    List G      (The encoding table)

8448 49 4A 4B 4C 4D 4E 4F  50 51 52 53 54 55 56 57
   A  Z  B  Y  C  X  D  W   E  V  F  U  G  T  H  S
  41 5A 42 59 43 58 44 57  45 56 46 55 47 54 48 53

8458 59 5A 5B 5C 5D 5E 5F  60 61 62 63 64 65 66 67
   I  R  J  Q  K  P  L  O   M  N  N  M  O  L  P  K
  49 52 4A 51 4B 50 4C 4F  4D 4E 4E 4D 4F 4C 50 4B

8468 69 6A 6B 6C 6D 6E 6F  70 71 72 73 74 75 76 77
   Q  J  R  I  S  H  T  G   U  F  V  E  W  D  X  C
  51 4A 52 49 53 48 54 47  55 46 56 45 57 44 58 43

8478 79 7A 7B 7C 7D 7E 7F  80 81 82 83 84 85 86 87
   Y  B  Z  A           1  3   ▒  \  D  H  ▒  V  B  (
  59 42 5A 41 20 20 31 33  00 9C 84 C8 00 96 C2 28
```

```
DIRECTORY DF0: CYPHER     05     List H      BYTE  COUNT

ATR  NAME TYPE VR SBLK SIZE LBC LADR SADR  HEX     DEC

 03 CTE    .PRG;01 0005 0022 80  8200 8200  1100    4352
 03 CTA    .PRG;01 0027 002E 43  82A0 9000  16C3    5827
 03 IDA8   .PRG;01 0055 0040 80  8200 8200  2000    8192
 03 IDAE   .PRG;01 0095 0040 80  E000 E000  2000    8192
 03 MLDP   .PRG;01 00D5 003F 80  8200 8200  1F80    8064
 03 MLDP   .PRG;02 0114 003F 80  E000 E000  1F80    8064
 03 DBUG32.PRG;01 0153 0021 60  EB00 EB00  1060    4192
 03 SMP798.PRG;01 0174 0026 80  8200 8200  1300    4864
 03 SMP79A.PRG;01 019A 0026 80  AD00 AD00  1300    4864
 03 SMP79E.PRG;01 01C0 0026 80  ED00 ED00  1300    4864
 03 CYPHER.SRC;01 01E6 001C 56  0000 0000  0DD6    3542
 03 CYPHER.PRG;01 0202 0005 7E  8200 8200  027E     638
 03 CYPHER.PRG;02 0207 0005 7E  8200 8200  027E     638  <<<<
 01 <FREE SPACE>  020C 1000                0000       0
```

SOFTWARE CATALOG

The following information is reproduced by courtesy of Intelligent Computer Systems, 12117 Comanche Trail, Huntsville, AL 35803. (205-881-3800). It is extracted from their software catalog for CCII, 3650, and 9650 computers. Prices are current. To our knowledge, ICS is the only 'full line' East Coast distributor for CCII software. Personal experience has proved them to be conscientious, reputable, and knowledgable dealers. Colorcue recommends that you consider purchasing any of these software selections soon, because their availability in the future is uncertain.

The software catalog number appears first, followed by an evaluation code of six characters with the following significance:

PART NUMBER

SEPARATOR (' + ' indicates ROM version available.)

9 9 0 0 3 2 / Q D T F E C

User Experience Required
- A: Beginner
- B: Intermediate
- C: Advanced
- D: Expert
- X: Not rated

Available Disk Formats
- 1: CCII/3621 v6.78
- 2: CCII/3621 v8.79
- 3: 3651/9651 v9.80
- 4: 3651/9651 v9.80-3
- 5: Both 1 and 2
- 6: Both 2 and 3
- 7: Both 3 and 4
- 8: All types

Quality of Software
- A: Good
- B: Average
- C: Needs Improvement
- D: Inadequate
- X: Not rated.

Status of Documentation
- A: Good
- B: Average
- C: Needs Improvement
- D: Not available, not needed
- X: Not rated

Available Disk Types
- 1: 5'' MD for 3651/9651
- 2: 5'' CD for CCII
- 3: 8'' FD for 3651/9651
- 4: 5'' MD or 8''FD for 3651/9651
- 5: All disk types

Category of Software
- A: Business
- B: Finance
- C: Engineering
- D: Educational
- E: System Utilities
- F: Special Applications
- G: Games and Entertainment

BUSINESS

JH102A/AA58BA ASSEMBLY DATA BASE/$85. Assembly language program to write data fields into formatted SRC files; includes editing, sorting, and printing facilities. Change, delete, edit, review, sort, and search. Print entire record or up to five selected fields. Screen editor may edit files. Convert existing RND files to SRC. Fast disk access. Options available for extended performance.

JW129/AA58AA CARAMOR/$24.58. Car expense accounting with calculation of cost/mile, cost/day, etc. Can determine cost effectiveness of lease/rental/own vrs public transportation.

991545/AB58BA COMP-U-WRITER/$190. Full-feature word processor with single key commands, full screen editing, move, copy, delete, search and replace, boldface, underlining, selection of print parameters, fast disk access, mailmerge facility. Professional performance. (Colorcue is written on Comp-U-Writer. Ed.)

BG115 + AB25BA COMPUCALC/$120. ''Spreadsheet'' program in assembly with unique file recovery system, stores data on ARY files. Contains usual ''Visicalc''-like features. Efficient, trouble-free software.

EM122/BB58AA DECISION MAKER/$10. Program examines input parameters for best buy or optimum decision. Optional printer readout of input and output data.

EM111/AA58AA INVOICE & ORDER/$30. Write personallized invoices to printer, with inputs for discounts, taxes, shipping, credit card numbers; input error correction provided. Totals and customer ID can be stored on disk. Orders may also be written, with similar features, and from a data base for frequently-used vendors.

EM109/BD58AA LABEL/$15. Use any line printer to print address labels. Special print features of Epson and IDS printer families supported. Print individually or from disk file.

JW104/BA58AA PERT/$39.50. Create 7 PERT plans with 200 activities each on one disk. Create, modify and analyze plan to determine critical paths, with data accounting.

JH114 + AB58AA PLANNING CALENDAR/$17.50. Assembly language program to log 7 events/day to total of 400 events. Specify daily, weekly, monthly, quarterly, bi-annual, and annual repetition. Requires screen editor or assembly data base to set up files.

EM103/BD58AA SCHEDULE/$49.50. Generate, update, and save bar graph screen displays of scheduling. May be printed on Epson and IDS printer families using special character functions. Designed for multi-national aerospace projects and is readily applicable to many other complex situations.

EM120/BD58AA VU-GRAPH/$10. Quick generation of VU-Graphs for printout, with descriptor fields, centered text, highlighting.

991005/BA58CB BONDS/$20. See effects of bond price and yield by variables, use realtime dates or time-to-maturity, up to 5 call dates for callable bonds. Print amortization schedule with a single keypress.

991007/AA58BA EQUITY/$20. Depreciation by straight line, double-declining balance, constant percentage, sum of digits, and sink-fund methods. Capitallized cost of periodic changes of up to three assets simultaneously to determine best option. Solve for any variable in the capitalization equation.

EM202/BD58AB FINANCIAL PROGRAMS SERIES/$18.50. Twenty programs for daily use in business management, including future investment value, calculation of effective interest rate, earned interest table generation, depreciation rate, depreciation amortization, salvage value, principal loan, payment on loan, last payment on loan, term of loan, mortgage amortization, and more.

EM205/BD58AB GENERAL LEDGER/$48.50. Capacity is 99 credit and 99 debit accounts with 200+ records per month. Printouts include accounts summary report by month and quarter, quarter and year, and complete account report by item.

EM201/AD58AB GRAPHICS CHECKBOOK/$38.50. Accommodates 15 expense and 6 deposit categories. Display or print (with Epson or IDS) tables sorted by date, totals of one or all categories. monthly cash flow, totals for category, for month, for year.

JH206+AA58AB LEDGER PAD/$75. For business and home financial data. 80 rows by 32 columns, with titles, data editing, arithmetic computation, report generation (to screen and printer). Supports numbers to 99,999,999,99 and computes to 2 to the power 40 with $ 0.01 accuracy.

DP208/AA58AB PERSONAL BUDGET/$29.95. Enter and edit monthly budget items by category, show CRT summaries in tabular and plot form of income, expenses, and "net worth"(tabular form also on printer.)

991001/BB58BB PERSONAL FINANCE/$20. Calculate annuities, interest rates, and mortgage payments by several methods.

EM204/AD58AB REAL ESTATE INVESTMENT/$18.50. Calculate best mortgage, monthly payments, investment return, positive or negative cash flow.

DP207/AA58AB STOCK FUND SWITCH STRATEGY/$39.95. For telephone switch mutual funds (or similar) to determine maximum earnings. Favors switch to common stock during market rise, and to cash or money market on market fall.

ENGINEERING

AUG401/BD58AC ELECTRICAL ENGINEERING I/$10. Color graphics supported; Ohm's law for AC, DC circuits. Transformerless power supply design.

AUG402/BD58AC ELECTRICAL ENGINEERING II/$10. Color graphics supported; attenuator design for T, H, PI, and O pad configurations. Multivibrator timing design for astable and one-shot.

SP302/AB25AC GASMIL/$19.50. From mileage and gas purchase, computes last miles/gallon, average miles/gallon for last three fill-ups, and average miles/gallon to date. Data, by table or chart, to Epson MX-80 printer.

993004/CD58BC STATISTICS 1/$25. 1) FILES: generates, maintains and displays files for use by other programs; 2) REGRES: linear, log, exponential or reciprocal regressions with confidence limits and graph; 3) PLOT: plot one to three graphs on rectangular coordinates from disk file or equation; 4) STAT: compute measures of central tendency, dispersions, skews, movement about the mean, from grouped or ungrouped data; 5) GRAPH: display histograms or polygonial graphs from grouped or ungrouped data.

993006/CD58BC STATISTICS 2/$30. 1) FILES; 2) MLTREG: multiple linear regressions to 6 variables, with or without transformations; 3) POLREG: polynomial regressions to 5th degree; 4) DISREG: fits binomial, norm, or Poisson distribution to input data, plus CHI-square check for goodness of fit; 5) VARINZ: test several sets of data for variance, estimate evaluation mean.

993008/CD58BC STATISTICS 3/$30. 1) FILES; 2) TIMSER: smooth time series by trend regression and other cyclical procedures; 3) INDEX: computes eight types of index data for several sets of data; 4) CMPTIM: computes variation within or between sets of data; 5) RANK: rank analysis on data oairs using Mann-Whitney test, computes rank correlations.

EDUCATIONAL

BM310/AA25BD ASSEMBLY LANGUAGE TUTORIAL/$50. 18 lessons in assembly language programming.

992516/AA58BD BASIC LANGUAGE I/$25. Two disks, beginning tutorial.

992519/AA58AD BASIC LANGUAGE II/$20. Program algorithms, more advanced commands.

992512/AB58AG HANGMAN/$20. Word game. Disk also includes MATH TUTOR for simple arithmetic and logic, and TWO TO TEN, a simple card game for practice in sums.

992514/BB58AD MATH TUTOR/$20. Simple arithmetic. Disk also includes CHECKBOOK, for checkbook balance and income tax data; RECIPE for storage and retrieval of favorites; MATH DICE for practice of elementary arithmetic, and the BIORHYTHMS program from the Sampler.

AUG301/BD58AD PRIMES & PRIME FACT/$10. Prime numbers and factoring to primes.

SYSTEM UTILITIES

SYSTEM UTILITIES [Comments in this section by COLORCUE. Starred programs are highly recommended as basic software packages for both the CCII and 3650 series computers. Evaluations are made on the basis of completeness, reliability, versatility, and usefulness with all software versions. In some cases other software may perform as well for your needs. Only one recommendation is made in each category.]

Definitions: Screen editor: used to enter assembly language source code and any other text that can be stored in a SRC file (containing only ASCII characters). May function as a mini-word-processor. Printing capability to serial printer. Easy editing of text in full view on CRT. Will load and edit any SRC or TXT file, as well as any ASCII file with optional file extension type. Monitor: a program to examine and modify the 8080 registers in the course of program execution, sometimes single-step through a program, examine and modify memory. Assembler: a program to convert ASCII source code (mnemonics) into machine code, producing a memory image file, ultimately "runnable" as a finished PRG program. A good assembler finds coding errors and flags them, and produces hard-copy of assembly showing source code, machine code, label chart and address assignments. Debugger: an all-purpose tool for working with any kind of disk file, principally used to work with PRG files. A complete debugger will act as an interpreter, monitor, mini-assembler, and disassembler. It facilitates examination and repair of disk file damage, disassembly of ROM, error correction in BAS files.

JH855 + AA25CE ASSEMBLER/$20. Jim Helm's assembler with printer driver.

JH811/AB25BE BASE2/$15. Set printing options on BASE 2 printer.

** **QS816 + AB58BE 'The' BASIC EDITOR/$49.95.** Quality Software's tour-de-force.

CT851 + AA25AE BASMERGE/$19.95. Merge Basic programs.

JH856 + AA25BE BASSRC/$19.95. Basic code to SRC file conversion.

** **CT830 + XX25XE CLIST/$19.95.** List Basic programs to printer; converts control characters to printable symbols for increased accuracy of printed program. Works with more printers than most.

CT828/XX25CE COMTRX/$19.95. Terminal software, for network communications.

JH857 + AB25CE CROSS REFERENCE GENERATOR/$10. Generates cross reference of JMPs and CALLs from assembly code. SRC code for program available.

** **CT852 + AA25CE CTA-ASSEMBLER/$34.95.** Probably the best all-around assembler for ISC computers using FCS. Good printer driver, good error displays.

** **CT833 + AB58CE CTE-SCREEN EDITOR/$54.95.** Text move, copy, delete, control character inserion. Will accommodate upper-case-only if necessary; all keyboards. Printer driver not as good as CTA, but adequate. Designed for source code entry, may be used as limited word processor.

CT841/AA22BE DEBUG/$29.95. Editor, monitor, disassembler, mini- assembler. Fine debugging tool, supports serial printer.

CT825 + AA58AE DFM/CRC/$49.95. Disk duplicating; complete disk on 1 or 2 drives, single file to either drive. Scan disk by sector, protect files, change directory, write unique format.

JH838 + AA25BE DIRECTORY/$15. Creates directory SRC file compatible with Jim Helm's assembly database program.

CT822/AA25BE DIRMOV/$24.95. A directory handling system for up to 600 disk directories, with machine language sort by name or number; search, delete, add, save, or print results.

BG836/BD25BE DIRECTORY ORGANIZER/$19.50. Saves directory informaion in SRC file, for editing by screen editor.

JH837 + AB25AE DISK EDITOR/$20. Move, copy, rename disk files.

JH809 + AA58BE DRIVER/$10. Printer driver for SRC files, set printing parameters. Supports special control codes for Epson and Base2 printers.

JH815/AB25CE EXPANDED ASSEMBLER/$20. Printer driver, line overflow function, program count, auto-pause on error.

JH814/AA58BE EXPANDED SCREEN EDITOR/$20. Complete screen text editing with move, copy, delete block functions. Good programming. Not for 3651.

CT850 + AA58AE FILEMRGE/$24.95. Merge SRC files.

991527/AB47AE FORMATTER/$25. For 3651 5″ or 8″ disks.

SP845/AD25AE FORMATTER/$24.95. For CCII, includes disk drive speed monitor and MAZE game.

991532/AA58CE FORTRAN/$75. Fortran compiler, linker and library. Experienced users report satisfaction with this version of FORTRAN.

JH813 + AX58AE HEXDEC/$10. Hexadecimal/decimal memory dump of system or selected sections.

JH839 + AA25CE HEXDIS/$15. Assembly language disassembler, displays hex, object code, mnemonics to screen and printer. Creates SRC disk file is requested.

** **BG851/AA25CE IDA-Interpreter, Disassembler, Assembler/$49.50.**
This gets an entire page of stars. Software debugging tool. It is not a SRC code assembler in the usual sense, but does everything else to perfection. Fascinating graphics, unbelievable printer facilities, excellant monitor, absolutely written to ease debugging tasks. If you are a serious programmer, this one will do it all, better than you ever thought possible. Available for loading at 4000, 8200, A000, E000. [See W.S. Whilly's article, and review this issue.]

CT853/AA25CE LDA FILE/$19.95. Convert LDA files to PRG the easy way.

BM810/XA25CE LDIS/$29. Two-pass labelling disassembler.

JH806 + AB58AE LISTER/$10. List BAS files to printer. Screens out control codes.

CT829 + AB25AE LLIST/$19.95. List BAS files to printer, set printer parameters.

JH812/XB25BE LOAD76/$10. For BASE2 printer; loads and prints user-defined character fonts.

991539/AB58DE MACROASSEMBLER/$50. Similar to assemblers used by "professional" computers. Includes linking loader and cross-reference generator. Some impossible bugs but generally usable for most things. Difficult instructions.

SP842/AD25AE MX80 BLOCK GRAPHIC DUMP/$14.50. MX80 graphic dump of CCII screen.

CT854 + AA25CE NUBUG/$39.95. Software debugging tool. Improved version of DEBUG.

CT831 + AA58BE PRINT II/$19.95. Printer driver portion of Com-Tronics software. Fine driver; set parameters, pause.

CT847 + AA58AE RENUM/$19.95. BAS file renumbering.

JH805/AB25DE REPACK/$10. Removes REM statements from BAS files.

JH840 + AA58BE SCREEN EDITOR/$20. Requires deluxe keyboard, complete screen editing. Not for 3651.

CT844 + AA58AE SRCPRT/$24.95. Machine language dump of CCII screen to printer (especially IDS printers). Rests in high memory and 'on call' with key press. Requires very careful handling. Not satisfactory on 3651.

JH805/AB58BE SMERGE/$10. Merge SRC disk files.

CT831 + AA58BE SORT/$19.95. Binary-weighted sort program, callable from BASIC. Sorts up to 1000 variable length array elements. Rests in high memory or at 4000H. Reliable.

JH808/AB25CE SORT PROGRAMS/$20. Series of programs for SRC files; sort, set fields to equal length, add fields; add, remove, substitute, rearrange order of data. Versatile programming for SRC.

JH802/AB25CE SOURCE/$30. SRC file from PRG file. Select dump to printer and to disk file. Set printer parameters.

CT832 + AB25BE SRCBAS/$34.95. The reverse of BASSRC. Convert SRC files into executable BAS files. Good for modem users.

JH803/AB25CE STRIP/$10. Remove sections of SRC files to disk. Set up SRC code subroutine library.

CT846/AA25AE SUPER MENU/$19.95. Select programs from directory display by bar cursor.

BG801/AB58CE SUPER MONITOR/$30. Subset of IDA (see above) and really obsolete because of it. Good monitor and debugger, but 'how ya goin' ta keep 'em down 'n the farm...?

BG835 + AX58CE SUPER MONITOR PLUS/$49.50. Ditto.

**** CT821 + AA58BE TERM II/$69.95.** Terminal software. The early version seems more bug-free, the second is easier to be with visually. Fancy provisions for uploading and downloading, setting transmission parameters. Good writing, but can't someone get rid of the bugs? Ask ISC for both versions.

CT831/BB25AE VLIST/$19.95. List BAS file variables to screen or printer.

SJ843/AA25BE XEDIT/$25.50. 'A totally different screen editor' with programmable function keys. New program, but we've had no experience with it. Can someone write a review? It sounds interesting.

GAMES

990046/AB58AG AIRRAID/$20. Airraid, Car race, Rover robot and Tiles.

SR962/AD37AG ALIENS/$25. Good graphics, fast.

AUG930/BC58BC B747 FLIGHT SIMULATION/$15. New York to Hartford. We need a review!

JH908/AA58AG BIORHYTHM/$12.50. Assembly language with printout.

990040/AB58AG BLACKJACK/$20. Blackjack, Roulette, Drag race, Horse race, Slot machine.

990052/AA58AG BOUNCE/$20. Bounce, Battleship, Slither.

AUG934/AB58AG BOUNCE BALL/$10. Bounce ball, Crossword, Create crossword puzzle, and 2 person chess.

990036/BB58AG CHESS/$20. Chess, Acey-ducey, Line five, Biorhythms.

SB950/AA58AG CHOMP/$29.95. Assembly masterpiece by David Suits.

JH901/AA25AG CRIBBAGE/$20. A peg board on a screen.

990042/AB58AG CUBIC TIC-TAC-TOE/$20. Tic-tac-toe, Greed, Galaxy, Space lander.

RT960/AD25AG FINAL FRONTIER/$25. Rick Taubold's space battle with the Klingons.

AUG933/AD58AG FOOTBALL/$10. Football, Space colony, Drop the marble,15 piece puzzle, Mastermind.

JH937/AB25AG FOOTBALL/$20. Assembly language, with real time clock.

QS910/AD26AG INVADERS/$34.95. The 'arcade' version with sound, Tic-tac-toe, Battleship.

JH906/AD25AG KALEIDOSCOPE/$10. Assembly language graphics thriller.

JH905/BB25AG LIFE/$10. Assembly language life and death struggle.

990056/AB58AG LUNAR LANDER/$20. Lunar lander, Coalition, Linko.

PS952/AD58AG MAZE/LUNAR LANDER/$14. Help a mouse find the cheese, and David Suits land on the moon.

990060/AB58AG MAZE MASTER/$20. Maze, Crossword puzzle, Create crossword.

BG936/BD25AG MILLEBORNES/$15. Complete version of the French card game.

AG935/AD58AG MONOPOLY/$10. Monopoly, Maze, Puzzle, and Hyper-space war game.

990034/AB58AG OTHELLO/$20. Othello, Math dice, Concentration.

WL949/AB25BG PROJECT APOLLO/$14.50. Dock Apollo on Skylab. Good graphics.

JH904/AB25AG PYRAMID SOLITAIRE/$15. The card game for the recluse.

SR961/AD37AG ROBOT WARS/$25. Fast action space game.

AUG932/BD58AG ROULET/$10. Roulet, Reverse number game, Dog chase cat, Ask Eliza, Clock, Robot.

JH907/AD58AG SCROLL/$10. Scrolling demo with source code.

990044/AB58AG SHARKS/$20. Sharks, Towers, Kalah, Mill.

990054/AB58AG SHOOT/$20. Shoot, 15 piece puzzle, Hyper-space war game, Seawar.

GH947/AD25AG SNAKES & LADDERS/$25. Board game from Australia with sound. Lemonade stand, Time signature.

JH903/BB25AG SOLITAIRE/$15. Standard solitaire.

JP940/AD25AG SPACE ARCADE I/$29.50. Three fast-action games: Galaxy attack, Invaders, Galactic patrol.

JP941/AD58AG SPACE ARCADE II/$29.50. Meteor mission, Orbits, Star battle.

AUG931/CD25AG SPACE WAR/$10. Good graphics.

GH946/AD25AG STAR FIGHTER/$20. Games with sound, plus word puzzle.

BJ920/BD25AG STAR TEC/$19.95. Star trek, Mastermind, Bounce, and x-y graph plot demo.

9048/AB58AG STAR TRADER/$20. Star trader, Color hunt, Decision maker, Calendar, Concentration.

990050/AB58AG SWARMS/$20. Swarms, Human reaction time, Roulette, Reverse numbers, Captain Alien.

SOFTWARE REVIEW

Joseph Norris

The primary source of software for the Compucolor/Intecolor line of computers is Intelligen Computer Systems, in Huntsville, Alabama, whose catalog is printed in this issue. In addition, there are several other sources of programs, source code, and programs in ROM, some sources with only a few offerings. This summary will treat these additions to the software library. It is important, too, to remember the disk libraries of the several user groups. Catalogs of these holding are available to members.

FASBAS. $25.00. Peter Hiner, 12 Pennycroft, Harpenden, Herts AL5 2PD, England. This is the one and only Basic Compiler for the CCII and 3651 (all versions.) At this time, all Basic commands are compilable. Peter is contantly making improvements, which are available to previous buyers free of charge if they send back the disk (or send $5 for a new disk.) A new compiler, called ZIP, is in preparation which Peter says makes Basic color graphics nearly as fast as assembler graphics. We'll give more details when ZIP is ready. The price of FASBAS is mine, not Peter's. It is approaching the obscene to pay him less for all his incredible effort. Don't you agree?

COLOR GRAPHICS DISKETTE. $20.00. Joseph Charles, PO Box 750, Hilton, NY 14468. This diskette contains almost all the demonstration programs listed in David Suit's book on color graphics for the CCII/3651 computers. It will save many hours of typing. When ordering, please specify the software version desired.

COLORWORD, word processor, screen editor. $50.00. (Shipment by airmail included.) Program Package Installers, PO Box 37, Darlington, Western Australia 6070. For CCII (all versions), and 3651 computers. Assembly program with 20K buffer for any keyboard. Will process existing SRC files (or any file extension representing an ASCII text file), with conventional word processing facilities: word wrap; block copy, move and delete; string search with optional replace; upper or lower case character set accommodated; HELP facility; printer parameter setup; automatic repeat on keystrokes; imbedding control characters for printer parameters; screen preview of printout; compact FCS file storage; all FCS commands available.

COM-TRONICS. Com-Tronics software is available from Inteligent Computer Systems. There is no current activity for CCII from Com-Tronics, and they have not answered my mail. Fortunately, little software support would be required. Com-Tronics programs, by and large, work very well on all software versions. CTE and CTA are still my standards.

GARY DINSMORE. Selected software available. See advertisement in last Colorcue. Gary is still interested in establishing an independent software marketing arm for CCII. Write to him for details.

BILL GREENE. Bill has current releases of Compucalc, IDA, some games, and a (free) version of FORTH for the CCII. His software is available through Intelligent Computer Systems. FORTH may be obtained by writing to Bill (or to Colorcue for a version of FORTH for 3651. Colorcue will also assist in providing 3651 versions of any other Greene software.)

JIM HELMS. Jim has announced that he will not support his software for the CCII/3651 beyond October of 1984. The source code for his software is therefore for sale at a cost equal to 1.5 times the selling price of the assembled version. Source code may be ordered from Jim Helms, and purchasers must agree not to distribute such purchased code. User groups may purchase source code for their libraries, with distribution regulated by the same rules that apply to disk library holdings. Software compatibility to 3651 computers has been a problem for Helm software. If versions dated 1983 or later are used, they may be assumed compatible with 3651. Otherwise, Jim cannot guarantee results.

Helms software is available from Jim Helms or from any of these authorized dealers: Intelligent Computer Systems, 12117 Comanche Trail, Huntsville, Alabama 35803; Howard Rosen, PO Box 434, Huntingdon Valley, PA 19006; Program Package Installers, PO Box 37, Darlington 6070, Australia. Some price reductions have been made recently. Note that many programs are available in ROM for use at 4000H.

Jim Helms. 1121 Warbler, Kerrville, TX 78028. (512-895-5136)

METIER. This software firm specializes in programs for educational use in schools. They write software to specification in Basic for a variety of computers, including the CCII and 3651. Most software may be altered to suit specific needs. Purchaser of programs must agree to the sale contract prohibiting copying or redistributing the materials. Some currently-available programs are listed here:

Database - Names, Addresses, Phone numbers (home use) $27.95

Programs for Trigonometry, $29.95

Names, Addresses, Student Data (for teachers), $27.95. (Note: the above programs may be purchased by Colorcue subscribers for $15.00 for a limited time.)

Budgets, Requisitions and Cash Flow (useable with Title IV grants), $700.00.

Please write to METIER for a complete description of this software and other services.

METIER. PO Box 51204, San Jose, CA 95151.

PROGRAM PACKAGE INSTALLERS. This Australian firm is offering selected software on disk or in ROM (for 4000H):

Colorword, $55; General Ledger, $70; Disk Editor/Formatter, $20; Directory (for use with Database II), $20; WISE-

II (emulator), $25; Lister (Basic programs), $15; Printer Driver (for SRC files), $15; Screen Editor and Assembler, $30; Cross Reference Assembler Utility, $15; BASSCR, HEX/DIS, SOLITAIRE, and CRIBBAGE, $15 each; 'THE' Basic Editor (available soon.)

PPI will copy your program to ROM for $10.00.

Program Package Installers. PO BOX 37, Darlington 6070, Australia.

Compucolor/Intecolor User Groups

The active user groups still serving Compucolor and Intecolor owners are listed here. As usership declines, there tends to be a collaborative spirit among these groups, sharing library holdings as well as articles for publication. Should you join more than one? If you are hungry for every bit of available information, I think so, particularly CHIP, CompUKolour, and CUVIC. These publications will be particularly helpful in providing the little bit of hardware source material that might be available. There are borrowings from material in Colorcue, and you may expect this to continue. However, local activity is still in evidence, and this activity is sometimes only available in the local publication. We have done our best to gather complete information. Please write to the group of your interest for more details.

AUSTRALIA

CUVIC, Victorian Compucolor Intecolor User Group. Monthly newsletter, about six pages, with meeting notes, reviews, software and hardware articles. Recent article titles have been on Comp-U-Writer techniques, reading and writing FCS files to Comp-U-Writer (including SRC and TXT files!), a Computerized Star Map, and conversion of disk drives for use with the CCII. I find CUVIC a good source of 'hard-to-find' material. Meeting attendance varies between 15 and 20. Total membership unknown. Subscription rate unknown, but probably about $30 for US readers. CUVIC maintains a disk library, $5 disk cost for copies of their holdings (plus postage for U.S.)

President: Ken Winder, 8 Brindy Crescent, East Doncaster, 3030, Victoria, Australia.
Secretary/Treasurer: Ted Stuckey, Box 420, Camberwell, 3124, Victoria, Australia.
Editor: Barry Holt, 19 Woodhouse Grove, Box Hill North, 3129, Victoria, Australia.
Address library requests to the Secretary.

CUWEST, Western Australia Compucolor Intecolor User Group, quarterly newletter, about 8 pages. Membership about 30. Articles in last issue included a discussion of Australia's VIDEOTEX system, and a reprint of Gary Dinsmore's article from Colorcue.
Secretary: John Newman, PO Box 37, Darlington, 6070, Western Australia. (Program Package Installers)

UPDATE, the publication of the Compucolor User Group of New South Wales. I have not seen this newsletter, but I suspect it serves a small membership.
Editor: Tony Lee. 52 Cowane Road, St. Ives 2075, New South Wales, Australia.

(Continued on back cover)

Back issues of COLORCUE contain a wealth of practical information for the beginner as well as the more advanced programmer, and an historical perspective on the CCII computer. Issues are available from October 1978 to current.

DISCOUNT: For orders of 10 or more items, subtract 25 % from total after postage has been added. **POSTAGE:** for U.S., Canada and Mexico First Class postage is included; Europe and South America add $1.00 per item for Air Mail, or $ 0.40 per item for surface; Asia, Africa, and the Middle East add $ 1.40 per item for Air Mail, or $ 0.60 per item for surface. **SEND ORDER** to Ben Barlow, 161 Brookside Drive, Rochester. NY 14618 for VOL I through VOL V; and to Colorcue, 19 West Second Street, Moorestown, NJ 08057 for VOL VI and beyond.

1978	**VOL I**	**$3.50 each**
	No. 1-3:	OCT/ NOV/ DEC
1979	**VOL II**	**$3.50 each**
	No. 1-3:	APR/MAY/JUN
	No. 4-5:	JAN/FEB/MAR
	No. 6-7:	AUG/SEP/OCT
	No. 8:	NOV XEROX COPY, $2.00
1980	**VOL III**	**$1.50 each**
	No. 1	DEC/JAN
	No. 2:	FEB
	No. 3:	MAR
	No. 4:	APR
	No. 5:	MAY
	No. 6:	JUN/JUL
1981	**VOL IV**	**$2.50 each**
	No. 0:	DEC/JAN
	No. 1:	AUG/SEP
	No. 2:	OCT/NOV
1982	No. 3:	DEC/JAN
	No. 4:	FEB/MAR
	No. 5:	APR/MAY
	No. 6:	JUN/JUL
	VOL V	
	No. 1:	AUG/SEP
	No. 2:	OCT/NOV
1983	No. 3:	DEC/JAN
	No. 4:	FEB/MAR
	No. 5:	APR/MAY
	No. 6:	JUN/JUL
1984	**VOL VI**	**$3.50 each**

GREAT BRITAIN

CompUKolour, the magazine of the Compucolour User Group (UK), published quarterly, or whenever, about 20 pages. Heavy article content and valuable material. Recents topics have included 'Improving your power supply', 'Transistor replacements', 'Compiling Basic' (Peter Hiner), and 'CREF, a cross-reference generator.' Issue 5 was dated December 1983, Issue 6 was dated March 1984. Membership fee is fifteen pounds (UK) yearly. Number of members unknown. This publication carries some resemblance to FORUM, and seems like a good source of materials. The user group maintains a disk library, and is currently sharing holdings with CHIP, in Rochester.

Secretary: Peter Hiner, 11 Pennycroft, Harpenden, Herts AL5 2PD, England.

Librarian: Bill Donkin, 19 Harwood Avenue, Bromley, Kent BR1 3DX, England.

Treasurer/Editor: John Booth, 27 Romany Lane, Tilehurst, Reading RG3 6AP, England.

UNITED STATES

CHIP, publication of the Rochester Compucolor Intecolor User Group. This venerable body, the granddaddy of us all, continues after six years as the leader in contributionship to periodicals, disk libraries, and general sustaining activity. Membership is $10 per year, which includes a subscription to CHIP (quarterly if lucky), and access to their very large disk library. I don't know their current membership, but monthly meetings draw from ten to twenty local members on a regular basis. Membership is international, however, and extensive. Newsletters can be 20 pages long. While content has been primarily a catalog of the disk library recently, CHIP, historically, is a source of material on all subjects relating to the CCII, hardware as well as software. An investment in back issues is well made (forgive me, Ben!). When the CCII is about to give it's final sigh, the Rochester User Group will most likely be there to ease it's last days. A remarkable collection of talent and dedication to whom we all owe a great deal. The list of names is a Who's Who for all former and current CCII materials. CHIP is currently working to assist 8000 users and investigate conversion of their library to 8000 format. CHIP maintains a library for the 3650 computer series as well. Write to them for more details.

Membership: Gene Bailey, 28 Dogwood Glen, Rochester, NY 14625

Librarian: Doug Van Putte, 18 Cross Bow Drive, Rochester, NY 14624

Editor: Ben Barlow, 161 Brookside Drive, Rochester, NY 14618

STAN PRO, West Coast user group and newsletter. I wish I could be more specific about this organization but they are, frankly, ephemeral at best. I have seen several newsletters, and they appear to include useful, and often 'rare' materials, with articles by names familiar and unfamiliar (Rick Manizar, Carl Hennig). Stan Pro has been an Intecolor dealer for some time, selling and servicing CCII, 3650 and 8000 series computers. Communications between Colorcue and Stan Pro have been less than satisfactory. The best information I have is this: the newsletter is still published quarterly(?), membership is about $35 per year, back issues are available (more than $100 per set). Service facilities are still available. Mr. Pro wrote to us recently giving permission to Colorcue to reprint articles from past issues of his newsletter. We will be purchasing the back issues and reviewing them. Thre is a large disk library associated with this group, perhaps 1000 or so programs. I know nothing about their content or availability. Perhaps you will want to communicate with Stan and see what is available.

S. P. Electronic Systems, 5250 Van Nuys Blvd, Van Nuys, CA 91401.

8000 SERIES COMPUTER USER GROUP

A relatively new user group for 8000 users is in the process of organization. I have had correspondence with several members and the outlook for interfacing with v6.78, v8.79 and v9.80-3 software looks promising. You may write to the 'founder', Glen Gallaway, at 1637 Forestdale Avenue, Beavercreek, OH 45432. Mr. Gallaway is in the process of moving and has a new address by this time, but perhaps his mail will be forwarded for awhile. Colorcue is interested in providing an outlet for articles, programs and source materials for the 8000. Please write if we can be of help. □

NEXT ISSUE: Last installment of Peter Hiner's series on Compiling Basic; W. S. Whilly masters the disk directory; Compucolor parts and service information; A switching box you can build; An 8000 article by Bob Mendelson....and more.